西安交通大学
软件学院

操作系统原理

Operating System Principle

田丽华

§7-5 银行家算法

Banker's Algorithm
银行家算法

银行家算法

- Multiple instances.（多个实例）
- Each process must a priori claim maximum use.
（每一个进程必须事先声明使用的最大量）
- When a process requests a resource it may have to wait.
（当一个进程请求资源，它可能要等待）
- When a process gets all its resources it must return them in a finite amount of time.
（当一个进程得到所有的资源，它必须在有限的时间释放它们）

Data Structures for the Banker's Algorithm
银行家算法的数据结构

Let n = number of processes, and m = number of resources types.
n为进程的数目，m为资源类型的数目

Available: Vector of length m. If available [*j*] = k, there are k instances of resource type *Rj* available.
（如果available[j]=k,那么资源Rj有k个实例有效）

Max: n x m matrix. If Max [i,j] = k, then process Pi may request at most k instances of resource type R_j.
（如果Max[i,j]=k,那么进程*Pi*可以最多请求资源*Rj*的k个实例）

Data Structures for the Banker's Algorithm
银行家算法的数据结构

Allocation: *n x m* matrix. If Allocation[*i,j*] = *k* then *Pi* is currently allocated *k* instances of *Rj*.

（如果Allocation[i,j]=k,那么进程Pj当前分配了k个资源Rj的实例）

Need: *n x m* matrix. If Need[*i,j*] = *k*, then P_i may need *k* more instances of *Rj* to compiete its task.

（如果Need[i,j]=k,那么进程*Pi* 还需要k个资源Rj的实例）

$$Need[i,j] = Max[[i,j] - Allocation[i,j]\ .$$

Banker's Algorithm

$Request_i$ = request vector for process P_i . If Requesti [j] = k then process P_i wants k instances of resource type R_j .

1. If $Request_i \le Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $Request_i \le$ Available, go to step 3. Otherwise P_i must wait, since resources are not available.
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

$$Available := Available - Request_i;$$
$$Allocation_i := Allocation_i + Request_i;$$
$$Need_i := Need_i - Request_i;$$

4. 用安全算法进行检查,看系统是否处于安全状态

If safe $\Rightarrow$ the resources are allocated to P_i .
If unsafe $\Rightarrow$ P_i must wait, and the old resource-allocation state is restored

Safety Algorithm
安全算法

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively. Initialize（让Work和Finish作为长度为m和n的向量）

   Work := Available
   Finish [i] = false for i - 1,3, …, n.

2. Find and *i* such that both: （找到i）

   (a) Finish [*i*] = false
   (b) $Need_i \leq Work$
   If no such *i* exists, go to step 4.

3. *Work := Work + Allocation*$_i$
   Finish[i] := true
   go to step 2.

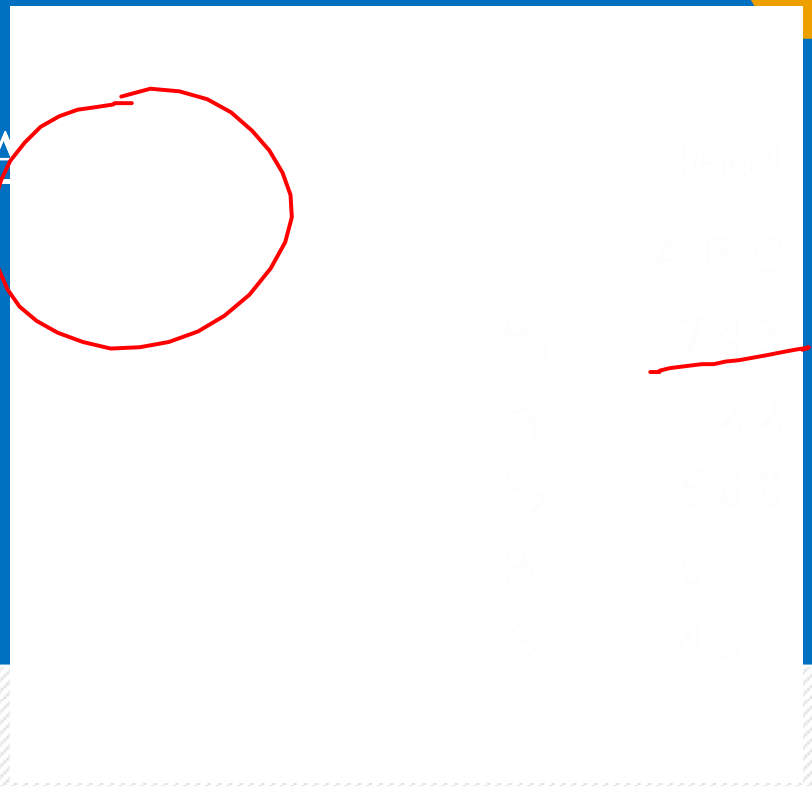
4. If Finish [*i*] = true for all *i*, then the system is in a safe state.

Example of Banker's Algorithm
银行家算法的例子

5 processes P_0 through P_4; 3 resource types A (10 instances), B (5instances, and C (7 instances). （5个进程P_0到P_4 :3个资源类型A(10个实例），B（5个实例），C（7个实例））

Snapshot at time T_0 :（时刻T0的片段）

	Allocation	Max	A
	A B C	A B C	
P_0	0 1 0	7 5 3	
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	



Example (Cont.)

用安全检测算法看能否找到一个安全序列

- Work[]=available=(3,3,2)
- Finish[i]=false (i=0..4)

	Work	need	allocation	finish
P1	3 3 2	1 2 2	2 0 0	T
P3	5 3 2	0 1 1	2 1 1	T
P4	7 4 3	4 3 1	0 0 2	T
P2	7 4 5	6 0 0	3 0 2	T
P0	10 4 7	7 4 3	0 1 0	T

- 存在安全序列： (P1，P3，P4，P2，P0)
- 安全 序列有时不是唯一的

Example (Cont.)

P1 request (1,0,2)

例子续

- Check that Request ≤ Available (that is, (1,0,2) ≤ (3,3,2) ⇒ *true*.
  (检查请求小于有效（就是说，(1,0,2) ≤**(3,3,2)**为真)

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

2 0 0　　3 2 2

- Executing safety algorithm shows that sequence <P_1, P_3, P_4, P_0, P_2> satisfies safety requirement. （执行安全算法表明序列p1,p3,p4,p0,p2满足要求）