死锁检测和恢复

死锁检测 (Deadlock Detection)

- 容忍系统陷入死锁状态
- 设计死锁检测算法，供需要时调用
- 配套系统恢复的对策

死锁检测算法的数据结构

- ***Available*** - 长度为 *m* 的矢量

available [*j*] = *k* 表示 R_j 类型的资源共有 *k* 个实例可用

- ***Allocation*** - *n* x *m* 矩阵

Allocation[*i,j*] = *k* 表示进程 P_i 拥有 R_j 类型的资源 *k* 个

- ***Request*** - *n* x *m* 矩阵

行向量 $Request_i$ [*j*] = *k* 表示进程 P_i 发出申请，申请 R_j 类型的资源 *k* 个

死锁检测算法

1. 令 ***Work*** 为长度 *m* 的矢量， ***Finish*** 为长度 *n* 的矢量。初始值:

   (a) *Work* = *Available*

   (b)For *i* = 1,2, …, *n*, if $Allocation_i \neq 0$, then
   Finish[i] = false;otherwise, *Finish*[i] = *true*.

2. 选取满足如下条件的 *i*

(a) *Finish*[*i*] == *false*

(b) $Request_i \leq Work$

If 不存在这样的 *i* ， then go to step 4

死锁检测算法（续）

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

4. If *Finish*[*i*] == false, for some *i*, $1 \le i \le n$, then 系统处于死锁状态。而且进一步得出结论， if *Finish*[*i*] == *false*, then 进程 P_i 死锁了。

Algorithm requires an order of O(*m* x $n^{2)}$ operations to detect whether the system is in deadlocked state.

死锁检测算法示例

- 5 个进程 P_0 至 P_4； 3 类资源 A (7 个实例), B (2 个实例), 以及 C (6 个实例)
- 在 T_0 时刻有

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- 得安全序列 $<P_0, P_2, P_3, P_1, P_4>$ 。系统没有死锁

死锁检测算法示例（续）

- *进程* P_2 提交一个申请，申请 1 个 C 类资源

	Request		
	A	*B*	*C*
P_0	0	0	0
P_1	2	0	1
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

- 死锁状态 ？
  - 进程 P_0 正常执行，并且释放其资源
  - 但是没有足够资源令其它进程正常执行
  - 死锁啦！ 集合里包括进程 P_1, P_2, P_3, P_4

死锁恢复： Process Termination 途径

- 所有死锁进程全部杀出
- 每次只杀出一个死锁进程，直至系统脱离死锁状态

死锁恢复： Process Termination 途径

- 依照什么顺序杀出死锁进程 ？
  - 进程优先级
  - 进程已经占用了多长 CPU 时间，进程还需要多长 CPU 时间
  - 进程占用资源的总数
  - 进程还需要多少资源
  - 杀出多少进程才能脱离死锁状态
  - 交互进程还是批处理进程
  - 。 。 。 。 。 。 。 。 。 。 。 。

死锁恢复：Resource Preemption 途径

- 选取牺牲品 – 剥夺哪些进程资源，使代价最小
- Rollback – 进程回滚至前面的安全状态，占有资源、申请资源均有变化
- Starvation – 某个进程总是不幸被选中，可能引发饥饿


End