3.7.2 利用银行家算法避免死锁

最有代表性的避免死锁的算法是Dijkstra的银行家算法。起这样的名字是由于该算法原本是为银行系统设计的，以确保银行在发放现金贷款时，不会发生不能满足所有客户需要的情况。在OS中也可用它来实现避免死锁。

3.7.2 利用银行家算法避免死锁

1. 银行家算法中的数据结构

为了实现银行家算法，在系统中必须设置这样四个数据结构，分别用来描述系统中可利用的资源、所有进程对资源的最大需求、系统中的资源分配，以及所有进程还需要多少资源的情况。

(1) 可利用资源向量Available。

(2) 最大需求矩阵Max。

(3) 分配矩阵Allocation。

(4) 需求矩阵Need。

3.7.2 利用银行家算法避免死锁

2. 银行家算法

设$Request_i$是进程P_i的请求向量，如果Request i[j]=K，表示进程P_i需要K个R_j类型的资源。当P_i发出资源请求后，系统按下述步骤进行检查：

(1) 如果Request i[j]≤Need[i, j]，便转向步骤(2)； 否则认为出错，因为它所需要的资源数已超过它所宣布的最大值。

(2) 如果Request i[j]≤Available[j]，便转向步骤(3)； 否则，表示尚无足够资源，P_i须等待。

3.7.2 利用银行家算法避免死锁

(3) 系统试探着把资源分配给进程P_i，并修改下面数据结构中的数值：

Available[j] = Available[j] - Request i[j];

Allocation[i, j] = Allocation[i, j] + Request i[j];

Need[i, j] = Need[i, j] - Request i[j];

(4) 系统执行安全性算法，检查此次资源分配后系统是否处于安全状态。若安全，才正式将资源分配给进程P_i，以完成本次分配；否则，将本次的试探分配作废，恢复原来的资源分配状态，让进程P_i等待。

3.7.2 利用银行家算法避免死锁

3. 安全性算法

系统所执行的安全性算法可描述如下：

(1) 设置两个向量：① 工作向量Work，它表示系统可提供给进程继续运行所需的各类资源数目，它含有m个元素，在执行安全算法开始时，Work := Available；② Finish：它表示系统是否有足够的资源分配给进程，使之运行完成。开始时先做Finish[i] := false；当有足够资源分配给进程时，再令Finish[i] := true。

3.7.2 利用银行家算法避免死锁

(2) 从进程集合中找到一个能满足下述条件的进程：

① Finish[i]=false;

② Need[i, j]≤Work[j];

若找到，执行步骤(3)，否则，执行步骤(4)。

(3) 当进程Pi获得资源后，可顺利执行，直至完成，并释放出分配给它的资源，故应执行：

Work[j] = Work[j]+Allocation[i, j];

Finish[i] =true;

go to step 2;

(4) 如果所有进程的Finish[i]=true都满足，则表示系统处于安全状态；否则，系统处于不安全状态。

3.7.2 利用银行家算法避免死锁

4. 银行家算法之例

假定系统中有五个进程{P_0, P_1, P_2, P_3, P_4}和三类资源{A, B, C}，各种资源的数量分别为10、5、7，在T0时刻的资源分配情况如图3-15所示。

资源 / 情况 / 进程	Max			Allocation			Need			Available		
	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
										(2	3	0)
P_1	3	2	2	2	0	0	1	2	2			
				(3	0	2)	(0	2	0)			
P_2	9	0	2	3	0	2	6	0	0			
P_3	2	2	2	2	1	1	0	1	1			
P_4	4	3	3	0	0	2	4	3	1			

图3-15 T_0时刻的资源分配表

3.7.2 利用银行家算法避免死锁

(1) T_0时刻的安全性：利用安全性算法对T_0时刻的资源分配情况进行分析(如图3-16所示)可知，在T_0时刻存在着一个安全序列$\{P_1, P_3, P_4, P_2, P_0\}$，故系统是安全的。

资源 / 情况 / 进程	Max			Need			Allocation			Work+Allocation			Finish
	A	B	C	A	B	C	A	B	C	A	B	C	
P_1	3	3	2	1	2	2	2	0	0	5	3	2	true
P_3	5	3	2	0	1	1	2	1	1	7	4	3	true
P_4	7	4	3	4	3	1	0	0	2	7	4	5	true
P_2	7	4	5	6	0	0	3	0	2	10	4	7	true
P_0	10	4	7	7	4	3	0	1	0	10	5	7	true

图3-16 T_0时刻的安全序列

3.7.2 利用银行家算法避免死锁

(2) P_1请求资源：P_1发出请求向量Request1(1, 0, 2)，系统按银行家算法进行检查：

① $Request_1(1, 0, 2) \leq Need_1(1, 2, 2)$；

② $Request_1(1, 0, 2) \leq Available_1(3, 3, 2)$；

③ 系统先假定可为P_1分配资源，并修改Available，$Allocation_1$和$Need_1$向量，由此形成的资源变化情况如图3-15中的圆括号所示；

④ 再利用安全性算法检查此时系统是否安全，如图3-17所示。

3.7.2 利用银行家算法避免死锁

可以找到一个安全序列{P_1, P_3, P_4, P_0, P_2}，系统是安全的，可以立即将P_1所申请资源分配给它。

资源 / 情况 / 进程	Work			Need			Allocation			Work+Allocation			Finish
	A	B	C	A	B	C	A	B	C	A	B	C	
P_1	2	3	0	0	2	0	3	0	2	5	3	2	true
P_3	5	3	2	0	1	1	2	1	1	7	4	3	true
P_4	7	4	3	4	3	1	0	0	2	7	4	5	true
P_0	7	4	5	7	4	3	0	1	0	7	5	5	true
P_2	7	5	5	6	0	0	3	0	2	10	5	7	true

图3-17 P_1申请资源时的安全性检查

3.7.2 利用银行家算法避免死锁

(3) P_4请求资源：P_4发出请求向量$Request_4(3，3，0)$，系统按银行家算法进行检查：

① $Request_4(3，3，0)\leq Need_4(4，3，1)$；

② $Request_4(3，3，0)>Available(2，3，0)$，让$P_4$等待。

(4) P_0请求资源：P_0发出请求向量$Request_0(0，2，0)$，系统按银行家算法进行检查：

① $Request_0(0，2，0)\leq Need_0(7，4，3)$；

② $Request_0(0，2，0)\leq Available(2，3，0)$；

③ 系统暂时先假定可为P_0分配资源，并修改有关数据，如图3-18所示。

3.7.2 利用银行家算法避免死锁

(5) 进行安全性检查：可用资源Available(2，1，0)已不能满足任何进程的需要，故系统进入不安全状态，此时系统不分配资源。

资源 / 情况 / 进程	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	3	0	7	2	3	2	1	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			

图3-18 为P_0分配资源后的有关资源数据