華中農業大學
HUAZHONG AGRICULTURAL UNIVERSITY

计算机操作系统

倪福川
fcni_cn@mail.hzau.edu.cn

华中农业大学 信息学院

目 录

第一章 操作系统引论
第二章 进程的描述与控制
第三章 处理机调度与死锁
第四章 存储器管理
第五章 虚拟存储器
第六章 输入输出系统
第七章 文件管理
第八章 磁盘存储器的管理
第九章 操作系统接口
第十二章 保护和安全

第三章　处理机调度与死锁

3.1 调度层次和调度目标
3.2 作业与作业调度
3.3 进程调度
3.4 实时调度
3.5 死锁概述
3.6 预防死锁
3.7 避免死锁
3.8 死锁的检测与解除

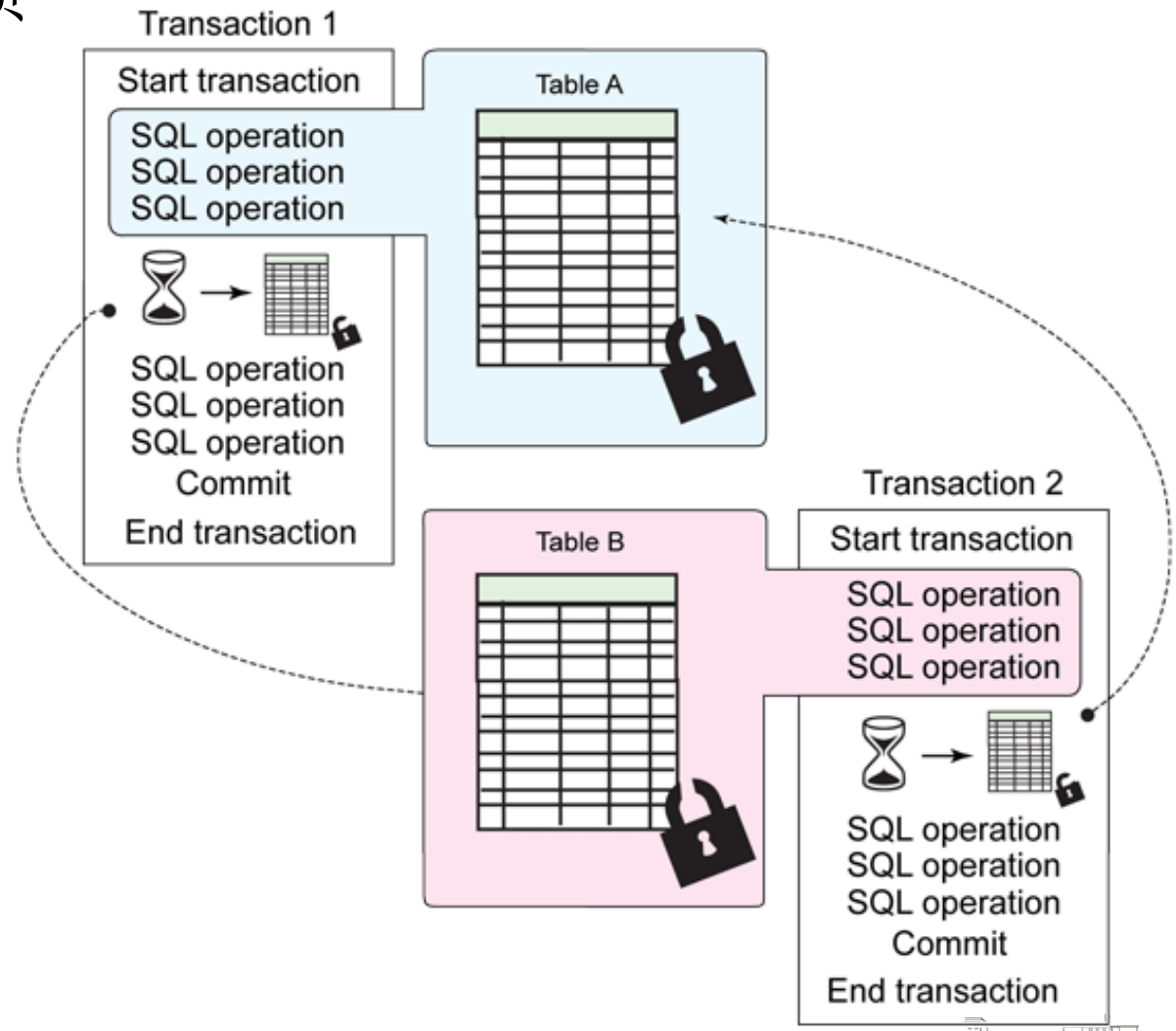
3.5 死 锁 概 述

3.5.1 资源问题

在系统中有许多不同类型的资源，可引起死锁的主要是，需要采用互斥访问方法的、不可以被抢占的资源，即在前面介绍的临界资源。

系统中这类资源，如打印机、数据文件、队列、信号量等。

数据库 死锁


Transaction 1
Start transaction
SQL operation
SQL operation
SQL operation
SQL operation
SQL operation
SQL operation
Commit
End transaction
Table A
Table B
Transaction 2
Start transaction
SQL operation
SQL operation
SQL operation
SQL operation
SQL operation
SQL operation
Commit
End transaction

1. 可重用性资源和消耗性资源

1) 可重用性资源

可供用户重复使用多次的资源：

(1) 只分配给一个进程使用，不允许多进程共享。

(2) 使用可重用性资源时，须按照特定的顺序：

① 请求资源。如失败，进程会被阻塞或循环等待。

② 使用资源。进程对资源进行操作；

③ 释放资源。当进程使用完后自己释放资源。

(3) 数目相对固定的，既不能创建也不能删除。

2) 可消耗性资源

临时性资源，在进程运行期间，由进程动态地创建和消耗的。

具有如下性质：

① 数目不断变化

② 进程可创造、增加可消耗性资源（放入该资源类的缓冲区）。

③ 进程可请求消耗可消耗性资源，不再返还 。

2. 可抢占性资源和不可抢占性资源

1) 可抢占性资源

某进程在获得这类资源后，该资源可以再被其它进程或系统抢占。 CPU 内存

2) 不可抢占性资源

一旦系统把某资源分配给该进程后，就不能将它强行收回，只能在进程用完后自行释放。 打印机

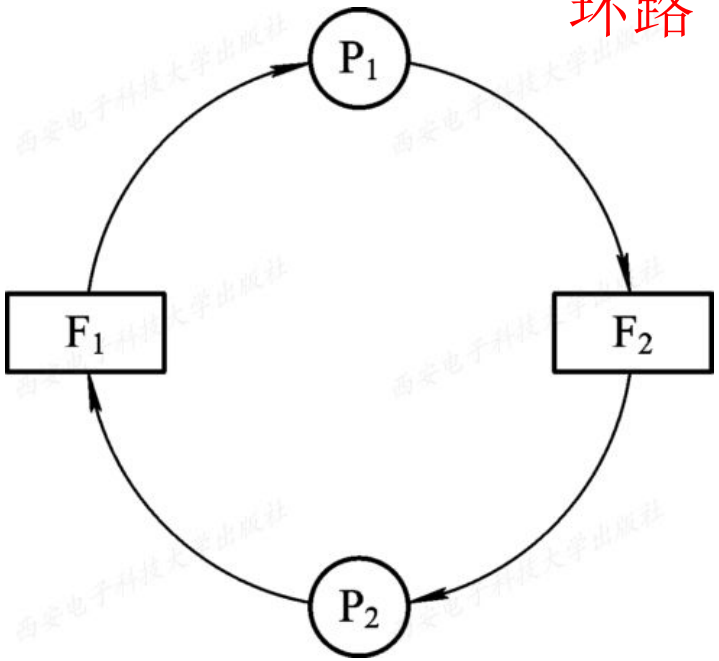
3.5.2 计算机系统中的死锁

1. 竞争不可抢占性资源引起死锁

通常系统中所拥有的不可抢占性资源其数量不足以满足多个进程运行的需要，使得进程在运行过程中，会因争夺资源而陷入僵局。

利用资源分配图进行描述，用方块代表可重用的资源(文件)，用圆圈代表进程，图3-12所示。

P_1	P_2
….	…
Open(f_1, w) ;	Open(f_2, w) ;
Open(f_2, w) ;	Open(f_1, w) ;

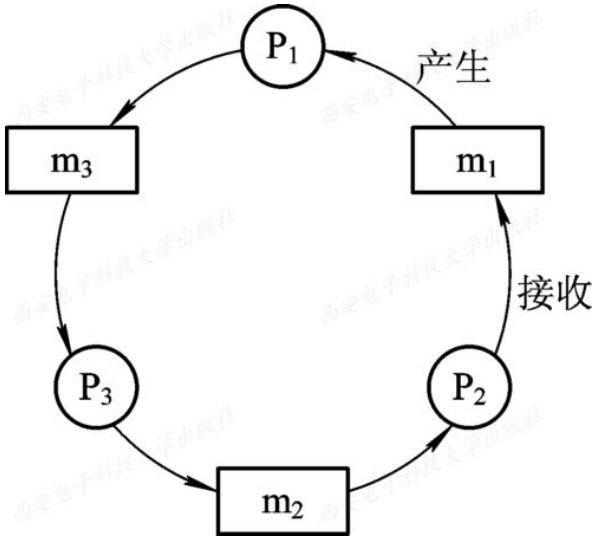


2. 竞争可消耗资源引起死锁

图3-13示出了在三个进程之间，在利用消息通信机制进行通信时所形成的死锁情况。

P_1: …send(p_2, m_1); receive(p_3, m_3); …
P_2: …send(p_3, m_2); receive(p_1, m_1); …
P_3: …send(p_1, m_3); receive(p_2, m_2); …

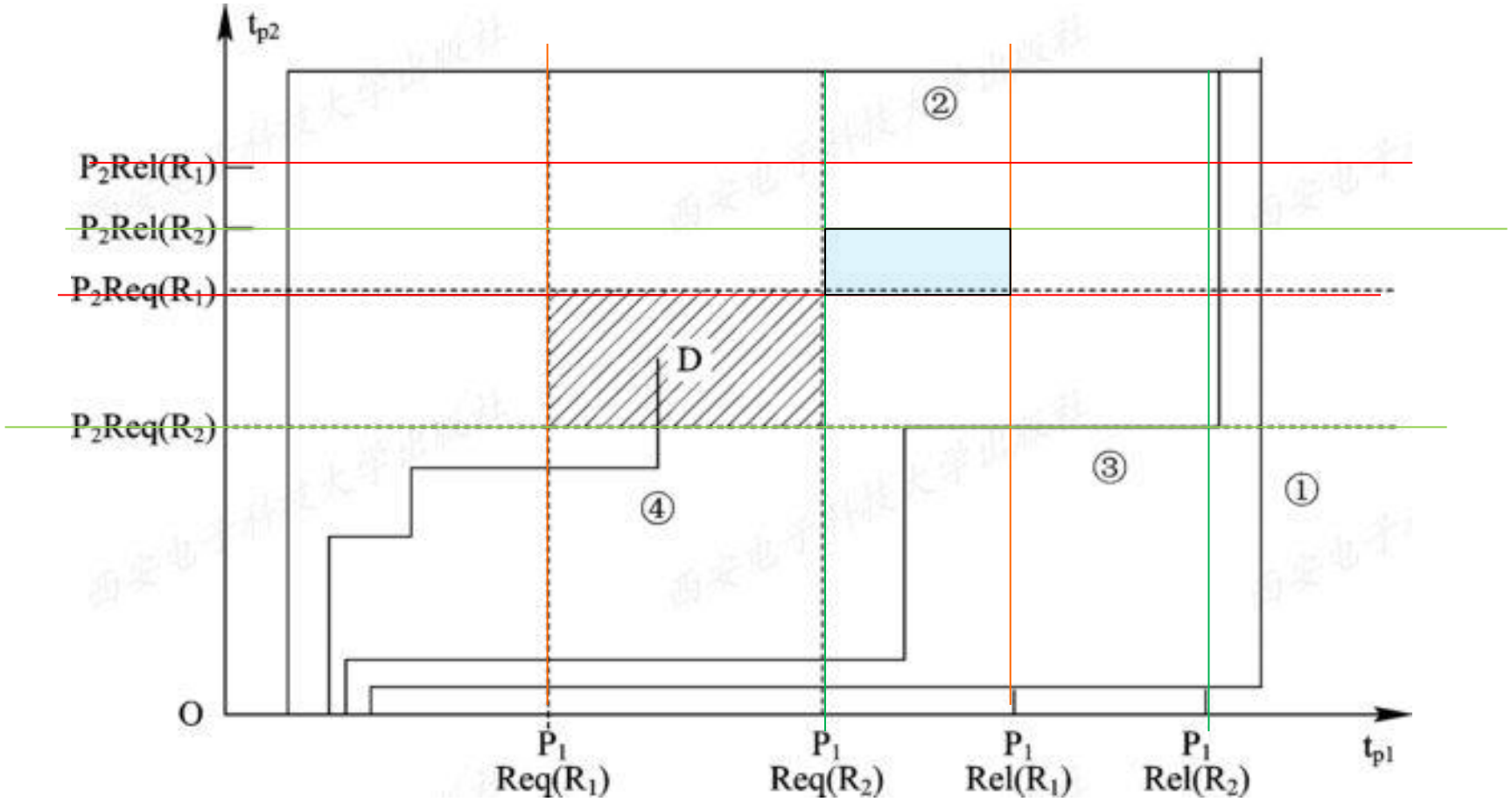
P_1: …receive(p_3, m_3); send(p_2, m_1); …
P_2: …receive(p_1, m_1); send(p_3, m_2); …
P_3: …receive(p_2, m_2); send(p_1, m_3); …



3. 进程推进顺序不当引起死锁

除了系统中进程对资源的竞争会引发死锁外，进程在运行过程中，对资源进行申请和释放的顺序是否合法，也可能产生死锁。

图3-14　进程推进顺序对死锁的影响



①：P_1：Request(R_1)→P_1：Request(R_2)→P_1：Releast(R_1)→P_1：Release(R_2)→P_2：Request(R_2)→P_2：Request(R_1)→P_2：Release(R_2)→P_2：Release(R_1)，

3.5.3 死锁的定义、必要条件和处理方法

1. 死锁的定义

P107

在一组进程发生死锁的情况下，这组死锁进程中的每一个进程，都在等待另一个死锁进程所占有的资源。

若一组进程中的每一个进程都在等待仅由该组进程中的其他进程才能引发的事件，那么该组进程是死锁的

2. 产生死锁的必要条件

产生死锁必须同时具备四个必要条件：

(只要其中任一个条件不成立，死锁就不会发生)

(1) 互斥条件。
(2) 请求和保持条件。
(3) 不可抢占条件。
(4) 循环等待条件。

互斥条件

指进程对所分配到的资源进行排它性使用，即在一段时间内某资源只由一个进程占有。

如果此时还有其它进程请求该资源，则请求者只能等待，直至占有该资源的进程用毕释放。

一个资源不允许两个或两个进程同时使用

请求和保持条件

指进程已经保持至少一个资源，但又提出了新的资源请求，而该资源又已被其他进程占有，此时请求进程阻塞，但又对自己已获得的其它资源保持不放。

不剥夺条件

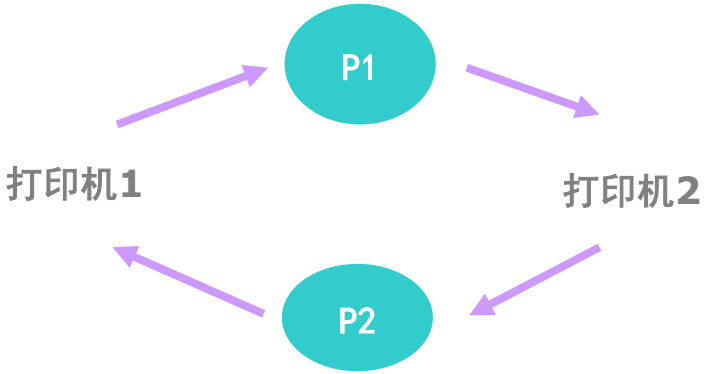
指进程已获得的资源，在未使用完之前，不能被剥夺，只能在使用完时自己释放。

注意区分与之前说到抢占优先级的区别，抢占优先级只是抢占进入就绪队列的位置，而不是直接占用了CPU，这里没有矛盾的

环路等待条件

指在发生死锁时，必然存在一个进程—资源的环性链：

即进程集合{P_0，P_1，P_2，—，P_n}中的P_0正在等待一个P_1占用的资源；P_1正在等待P_2占用的资源，—，P_n正在等待已被P_0占用的资源。



3. 处理死锁的方法

(1) 预防死锁。

(2) 避免死锁。

(3) 检测死锁。

(4) 解除死锁。

预防死锁

- 设置某些限制条件，破坏四个必要条件中的一个或几个条件。

想办法打破一个

- 简单、较易实现。

避免死锁

事先预防策略

- 在资源的动态分配过程中，用某种方法去防止系统进入不安全状态
- 可获得较高的资源利用率及系统吞吐量
- 实现上有一定的难度。

检测死锁

- 允许系统在运行过程中发生死锁。
- 设置的检测机构，及时地检测出死锁的发生，并精确地确定与死锁有关的进程和资源；
- 采取适当措施清除掉系统中已发生的死锁。

解除死锁

- 与检测死锁相配套。
- 当检测到系统中已发生死锁时，须将进程从死锁状态中解脱出来。
- 常用方法:撤消或挂起一些进程。

3.6 预 防 死 锁

破坏产生死锁的四个必要条件，以避免发生死锁。 互斥条件是非共享设备所必须的，应加以保证。

互斥条件 （x）

请求和保持条件 （√）

不剥夺条件 （√）

环路等待条件 （√）

3.6.1　破坏“请求和保持”条件

系统必须保证做到：当一个进程在请求资源时，它不能持有不可抢占资源。

两个不同的协议实现：

1. **第一种协议**：所有进程开始运行前，须一次性申请所需全部资源。

静态分配(预分配)

优点：

- 简单、易实现且安全

缺点

- 资源被严重浪费，恶化了系统的利用率；
- 饥饿，使进程延迟运行。

2. **第二种协议：**该协议是对第一种协议的改进，它允许一个进程只获得运行初期所需的资源后，便开始运行。

运行过程中，逐步释放已分配、且已用毕的资源，然后再请求新的资源。

- 提高设备利用率；减少饥饿机率

3.6.2 破坏“不可抢占”条件

已保持不可被抢占资源的进程，若请求新资源而不能得到满足时，必须释放已经保持的所有资源，待以后再重新申请。

进程已占有的资源会被暂时地释放，或说被抢占了，从而破坏“不可抢占”条件。CPU 内存

实现较复杂．代价大；反复申请和释放资源，进程执行无限推迟、延长进程周转时间，增加系统开销、降低系统吞吐量。

3.6.3 破坏“循环等待”条件

对系统所有资源类型进行线性排序，并赋予不同的序号；

所有进程请求资源严格按资源序号递增的次序提出，防止出现环路。

层次分配

摒弃“循环等待”条件的缺点

（1）序号须相对稳定，新设备类型增加受限。

（2）作业(进程)使用资源顺序与系统规定顺序不同而造成资源的浪费。

（3）限制用户编程。

设置

若系统中有m个资源被n个 进程共享，当每个进程都要求K个资源，而 [填空1] 时，如果分配不当，就有可能引起死锁。

作答

正常使用填空题需3.0以上版本雨课堂

3.7 避 免 死 锁

属事先预防策略，但事先并不限制产生死锁的必要条件，而在资源动态分配过程中，防止系统进入不安全状态，以避免发生死锁。

所施加限制条件较弱，可获得较好系统性能，常用来避免发生死锁。

3.7.1 系统安全状态

系统状态：为安全状态、不安全状态。

当处于安全状态时，可避免发生死锁。

当处于不安全状态时，则可能进入死锁状态。

允许进程动态地申请资源，但系统在进行资源分配前，应先计算此次资源分配的安全性。

P110

系统安全状态

系统能按某种顺序(P1，P2，…，Pn)，为每个进程分配所需资源，直至最大需求，使每个进程都可顺序完成。

若不存在这样一个安全序列，系统处于不安全状态。

系统进入不安全状态后可能进入死锁；

只要系统处于安全状态，便可避免进入死锁状态。

- 避免死锁实质：如何使系统不进入不安全状态。

例:假定系统有三个进程P_1、P_2和P_3，共12台磁带机。进程P_1要求10台磁带机，P_2和P_3分别要求4台和9台。假设在T_0时刻，进程P_1、P_2和P_3已分别获得5台、2台和2台磁带机，尚有3台空闲未分配，如下所示：

进　程	最 大 需 求	已 分 配	可　用
P_1	10	5	3
P_2	4	2	
P_3	9	2	

判断系统在T_0时刻的安全性

①谁需用的少分配给谁；
②需求一样时，分配给能返回最多的；（分配给既可以完成的，又能释放出更多资源的；分配给最快可以完成的）

3. 由安全状态向不安全状态的转换

如果不按照安全序列分配资源，则系统可能会由安全状态进入不安全状态。

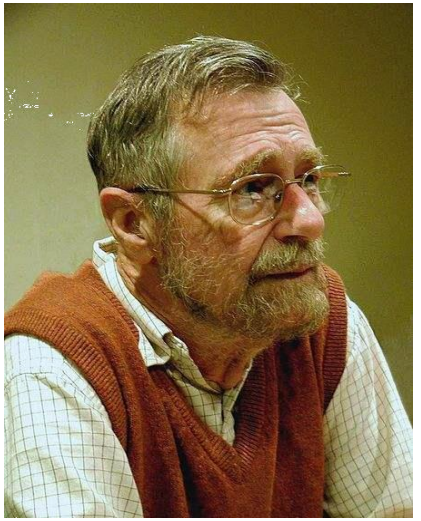
基本思想：

确保系统始终处于安全状态

3.7.2 利用银行家算法避免死锁

Dijkstra银行家算法

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。



1. 银行家算法中的数据结构
2. 银行家算法
3. 安全性算法
4. 银行家算法之例

信号量和PV原语
解决“哲学家聚餐”问题
图论：单源最短路径算法

1. 银行家算法中的数据结构

P112

(1) 可利用资源向量Available[j]=K。

系统中可利用的R_j类资源K个

(2) 最大需求矩阵Max[i，j]=K,

表示进程i需要R_j类资源的最大数值为K

(3) 分配矩阵Allocation[i，j]=K

进程i当前得R_j类资源的数目为K

(4) 需求矩阵Need[i，j]=K

表示进程i还需要R_j类资源K个，方能完成其任务。

Need[i，j] = Max[i，j] - Allocation[i，j]

2. 银行家算法

设Request $_i$[j]=K，进程P_i请求K个R_j类型的资源。系统按下述步骤进行检查：

(1) 如Request i[j]≤Need[i, j]，转向(2)； 否则认为出错；

请求资源数不能超过进程所宣布的需求。

(2) 如Request i[j]≤Available[j]，转向(3)； 否则，P_i须等待。

请求资源数小于等于系统能够提供的有效资源个数。

资源请求算法

(3) 试探把资源分配给进程P_i，并修改：

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) 执行**安全性算法**，检查此次资源分配后是否处于安全状态。

若安全，才将资源分配给进程P_i；否则，将试探分配作废，恢复原来的资源分配状态，让进程P_i等待。

3. 安全性算法

(1) 设置两个向量：

① 工作向量Work，可供n个进程继续运行所需的各资源数目，m个元素，初值Work := Available；

② Finish：表示系统是否有足够的资源分配给进程。初值Finish[i] := false；当有足够资源分配给进程时，再令Finish[i] := true。

(2) 从进程集合中找到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)。 $(n\times m)$

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

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

Finish[i] =true;

返回到(2); $((n-1)\times m + (n-2)\times m + \cdots + 1\times m)$

(4) 如所有Finish[i]=true，则处于安全状态；否则，处于不安全状态。

时间复杂度$O(m\times n^2)$

4. 例----银行家算法

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)：

（10、5、7）

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
P_1	3	2	2	2	0	0	1	2	2			
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			

（7、2、5）

(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(图3-16所示):

初始 Work := Available

10 5 7　　7 2 5　　3 3 2

①Need[i, j]≤Work[j];

②Work[j] = Work[j]+Allocation[i, j];

进程 \ 资源情况	work			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

T_0时刻存在安全序列{P_1, P_3, P_4, P_2, P_0}，故系统是安全的。

(2) P_1请求资源：P_1：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，资源变化情况如图3-15所示；

④ 安全性算法检查此时系统是否安全，如图3-17所示。

P_1：Request1(1, 0, 2)

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

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

10、5、7　　（7、2、5）　　3　3　2

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			

④ 安全性状态检查

2 3 0

①Need[i, j]≤Work[j];

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

③Work[j] = Work[j]+Allocation[i, j];

②Finish[i] =true;

图3-17 P_1申请资源时的安全性检查

存在安全序列{P_1, P_3, P_4, P_0, P_2}，故系统是安全的

可以将P1所申请的资源分配它

(3) P_4请求资源：$Request_4(3，3，0)$，

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。

10、5、7　　（8、2、5）　　2　3　0

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			

(4) P_0请求资源：$Request_0(0，2，0)$，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 系统暂假定可为P_0分配资源，并修改有关数据，如图3-18所示。

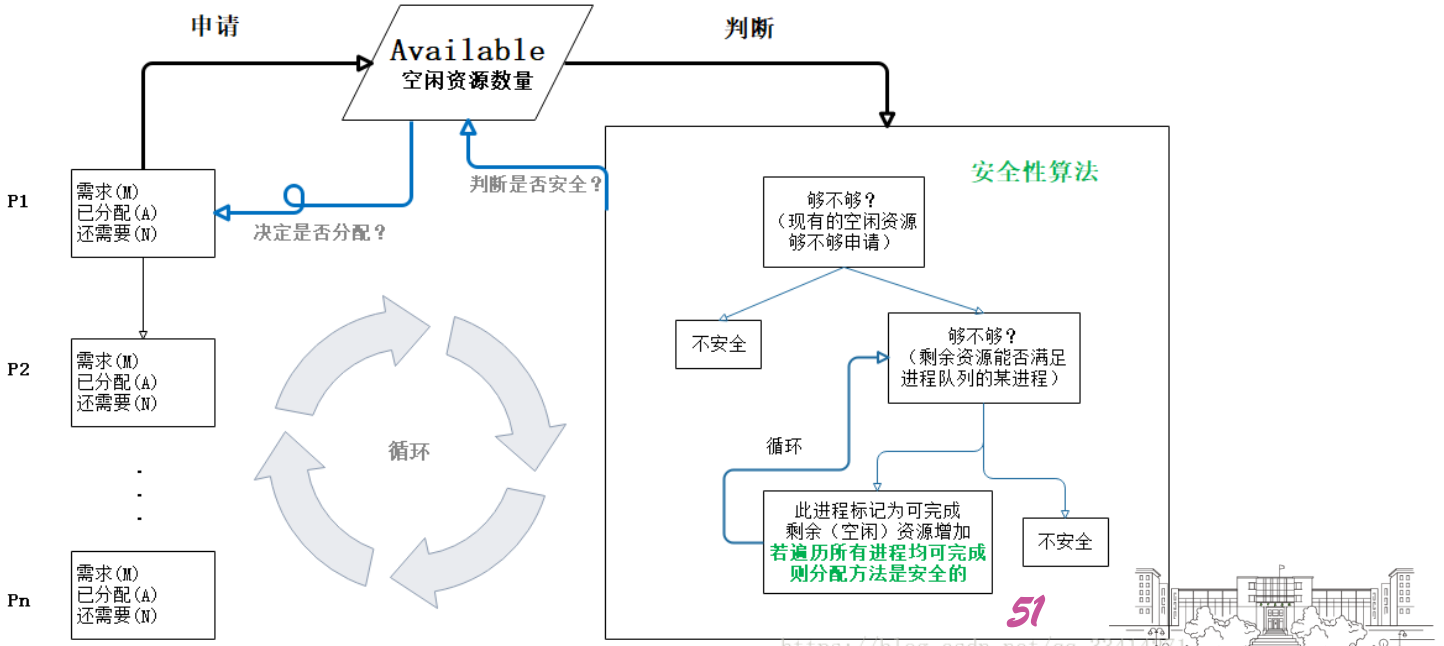
资源 / 情况 / 进程	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	3	0	7	2	3	2	1	0
P_1	3	0	2	0	2	0			
P_2	3	0	2	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

图3-18 为P_0分配资源后的有关资源数据

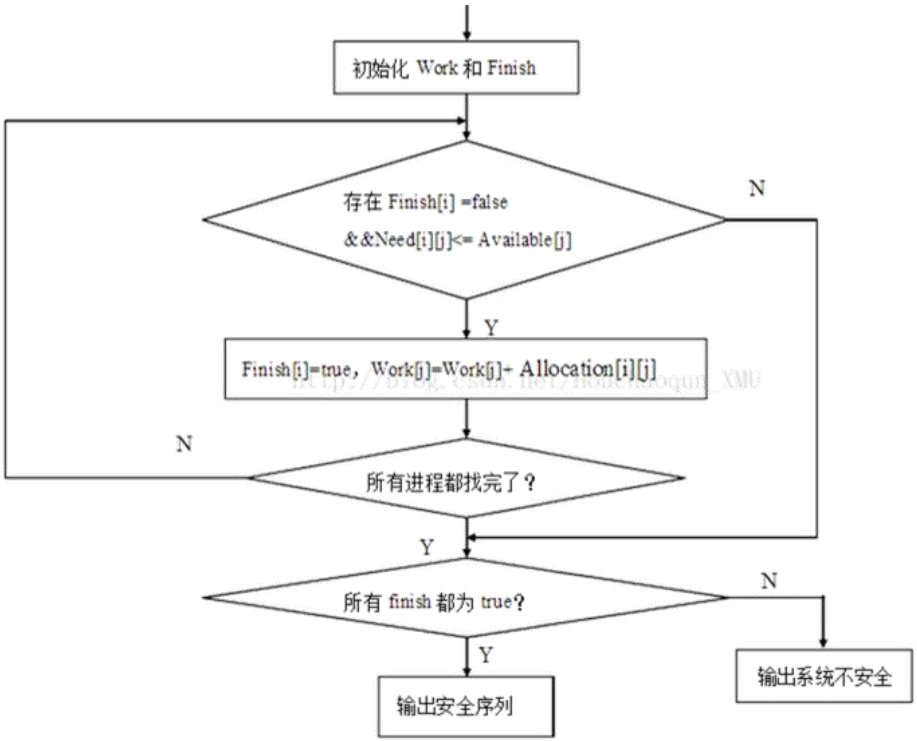
(5) 进行安全性检查：

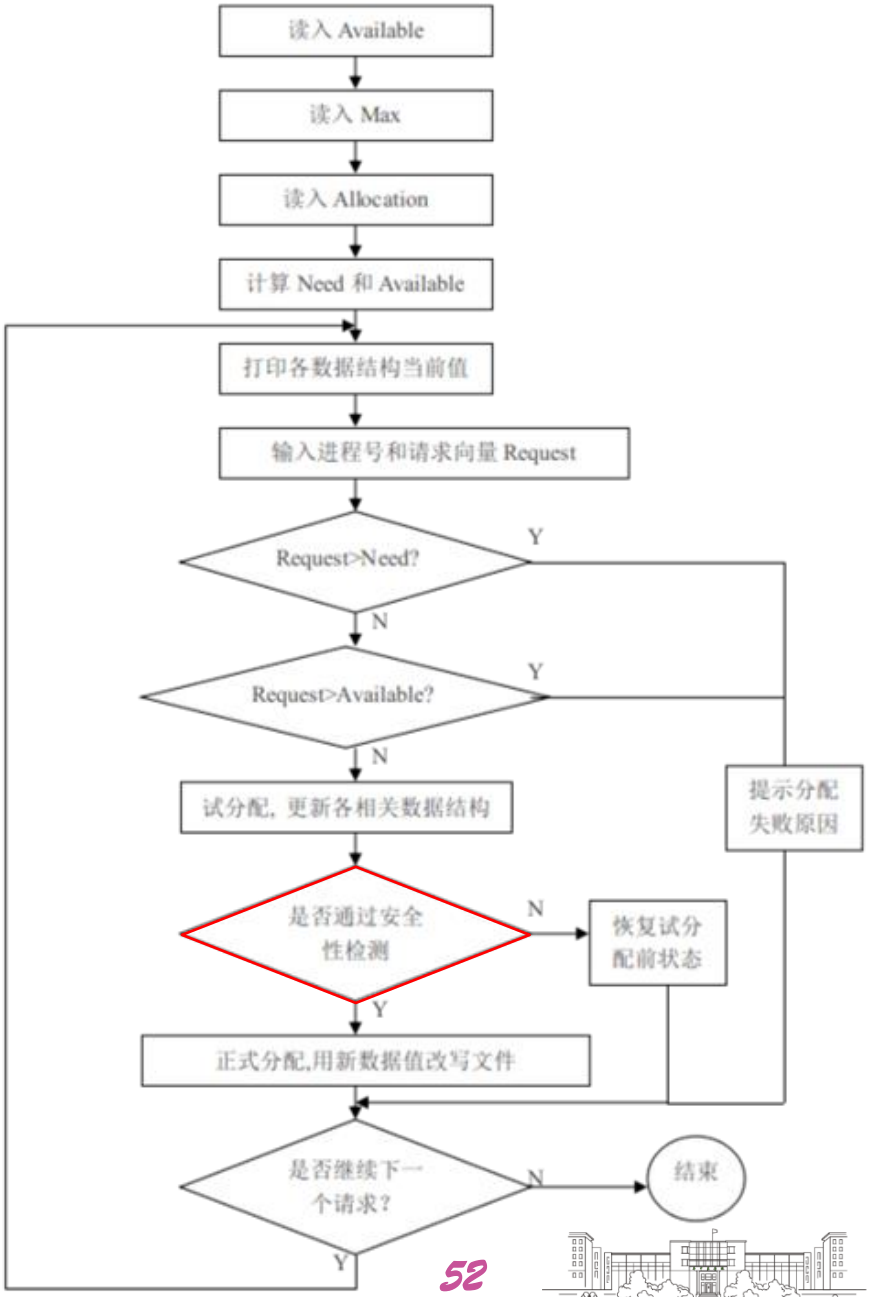
Work := Available；　　Need[i, j]>Work[j];

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。



代码实现


初始化 Work 和 Finish
存在 Finish[i] =false
&&Need[i][j]<= Available[j]
N
Y
Finish[i]=true，Work[j]=Work[j]+ Allocation[i][j]
N
所有进程都找完了？
Y
所有 finish 都为 true?
N
输出系统不安全
Y
输出安全序列


读入 Available
读入 Max
读入 Allocation
计算 Need 和 Available
打印各数据结构当前值
输入进程号和请求向量 Request
Request>Need?
Y
N
Request>Available?
Y
N
试分配，更新各相关数据结构
提示分配失败原因
是否通过安全性检测
N
恢复试分配前状态
Y
正式分配,用新数据值改写文件
是否继续下一个请求？
N
结束
Y

3.8 死锁的检测与解除

如果既不采取死锁预防措施，也未有死锁避免算法，系统很可能发生死锁。系统应提供：

① 死锁检测算法。用于检测系统状态，以确定系统中是否发生死锁。

② 死锁解除算法。当认定系统已发生死锁，法可将系统从死锁状态中解脱出来。

3.8.1 死锁的检测

为了能检测系统是否已发生死锁，必须：

① 保存有关资源请求和分配信息；

② 提供检测系统是否已进入死锁状态算法。

Warshall传递闭包

3.8.1 死锁的检测

可用资源分配图来描述死锁，

1. 资源分配图(Resource Allocation Graph)

对偶 G = (N, E)，由结点N和边E所组成：

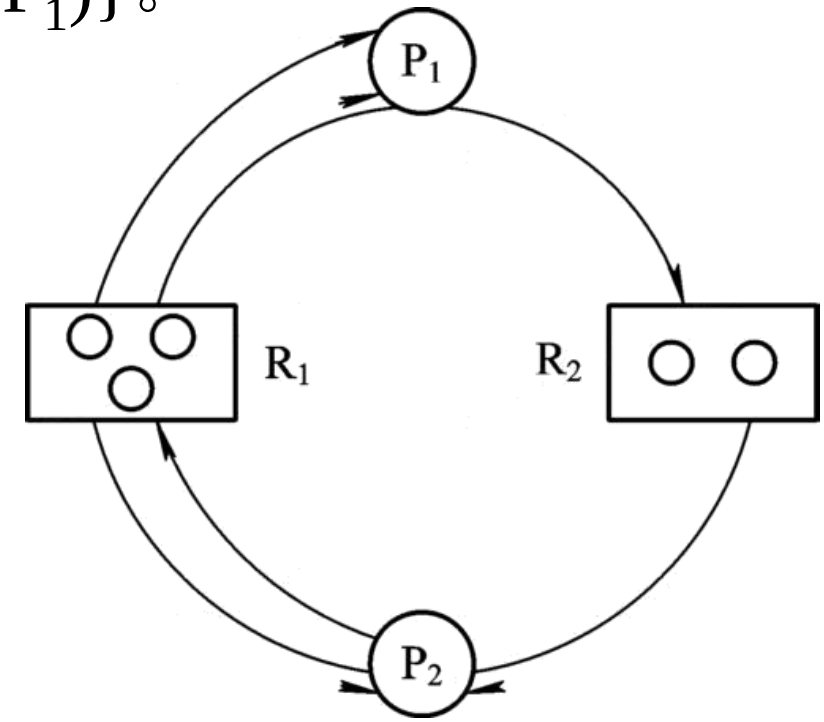
(1) N = P∪R ，P、R 互斥子集，即进程结点P={P_1, P_2, …, P_n}和资源结点R={R_1, R_2, …, R_n} 。

(2)资源请求边e∈E，e = {P_i, R_j}，表示进程P_i请求资源R_j；资源分配边E = {R_j, P_i}，表示把资源R_j分配给进程P_i。

P115

图3-19所示，G = (N, E)：

P = {P_1, P_2}，R = {R_1, R_2}，N = {R_1, R_2}∪{P_1, P_2}。两个请求边和两个分配边，即E = {(P_1, R_2), (R_2, P_2), (P_2, R_1), (R_1, P_1)}。



2．死锁定理

P116

用简化资源分配图来检测当系统状态是否为死锁状态。资源分配图简化方法：

(1) 找一既不阻塞又非独立的进程P_i，若P_i获得所需资源，运行完毕，释放所占全部资源，则相当于消去P_i请求边和分配边，使之成为孤立结点。

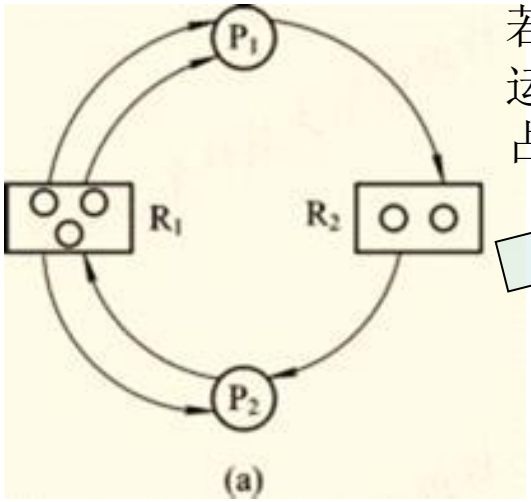
(2) P_i释放资源后，便可使P_j获得资源运行完成，又释放它所占有全部资源，即将P_j请求边和分配边消去。

(3) 反复对资源分配图简化，若能消去图中所有的边，使所有进程结点都成为孤立结点，则称该图是可完全简化的；若不能使该图完全简化，则称该图是不可完全简化的。

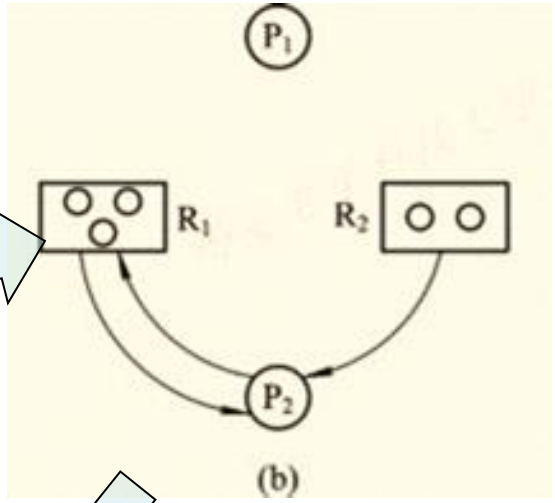
所有简化顺序都得到相同的不可简化图。

死锁定理：当系统处于S状态， S为死锁状态的充分条件是：当且仅当S状态的资源分配图是不可完全简化的。

资源分配图的简化



若P_1获得所需资源，运行完毕，释放所占全部资源



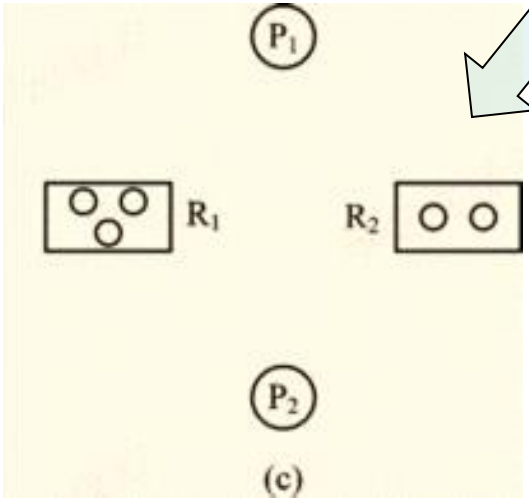


图3-20

可完全简化

3．死锁检测中的数据结构

类似银行家算法的数据结构：

(1) 可利用资源向量Available，每一类资源的可用数目， m类资源。

(2) L表: 不占用资源的进程集 (向量Allocation=0)

3．死锁检测

Work=Available;

L= {L_i |Allocation $_i$=0 ∩ Request $_i$=0}

L_i—> Finish[i]=false

不占用资源的进程记入L表

for (all Li∉ L)

{

for(all Request $_i$ ≤ Work) {

Work=Work + Allocation $_i$;

L_i ∪ L;

}

}

deadlock=(L={P_1, P_2, …, P_n});

乐观性假设

简化其资源分配图

时间复杂度$O(m \times n^2)$

3．死锁检测中的数据结构

(1) 可利用资源向量Available，m类资源中每一类资源的可用数目。

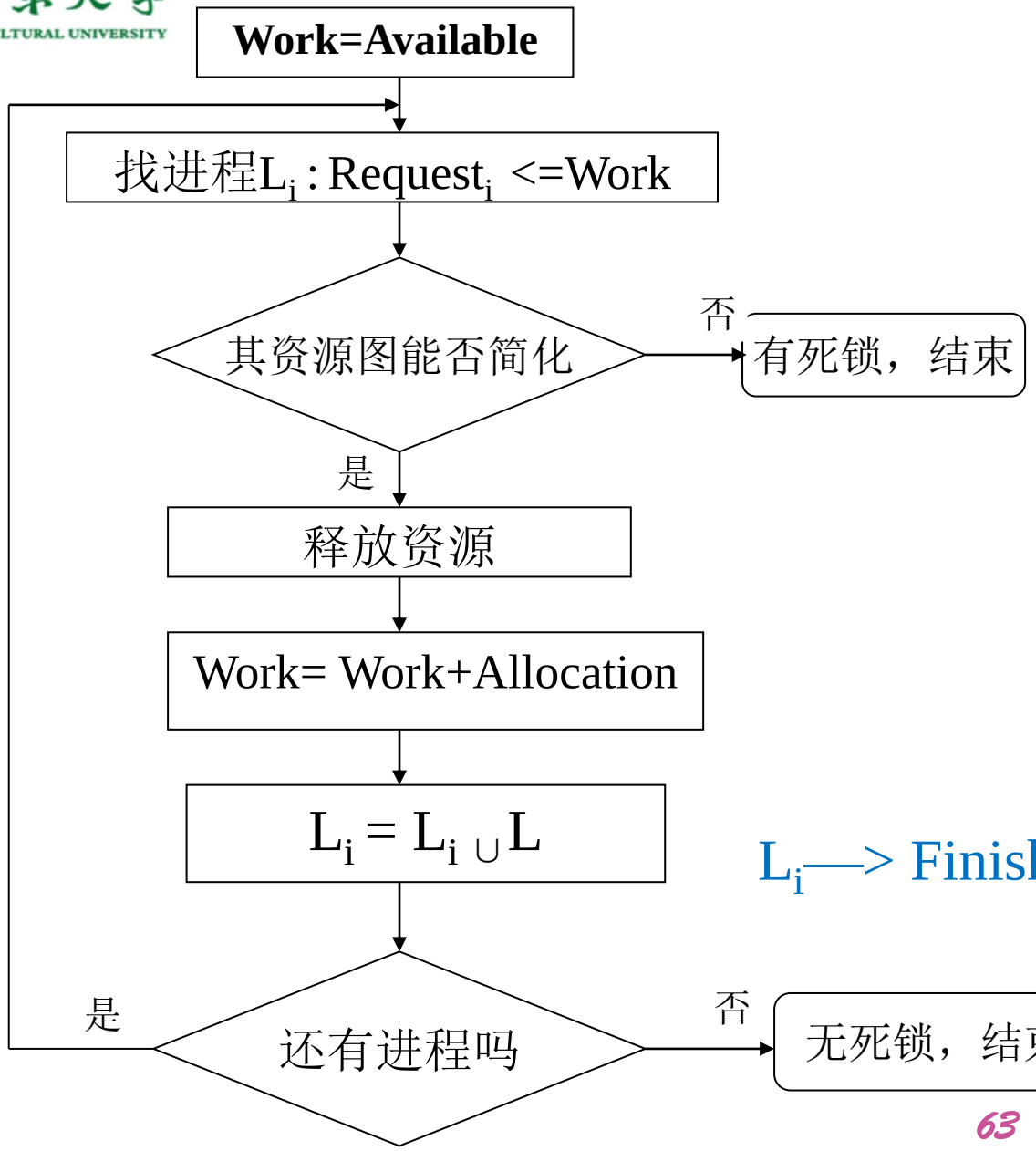
(2) 把不占用资源的进程(向量Allocation=0)记入L表中，即$L_i \cup L$。

(3) 从进程集合中找一个$Request_i \leq Work$的进程：

①简化其资源分配图，释放资源，增加工作向量Work =Work + Allocation i。

② 将它记入L表中。

(4) 若不能把所有进程都记入L表中，则状态S的资源分配图是不可完全简化，将发生死锁。



L_i—> Finish[i]=false

死锁检测

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7、2、6，在T0时刻的资源分配情况,判断系统状态：

	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			

$Request_2(0，1，0)$

3.8.2 死锁的解除

1. 终止进程的方法

1) 终止所有死锁进程

死锁自然解除，但代价大。

2) 逐个终止进程 稍微温和

按照某种顺序逐个终止进程，直至有足够资源，打破循环等待，把系统从死锁解脱出来。

选择策略:为死锁解除所付出的“代价最小”

P117

2. 付出代价最小的死锁解除算法

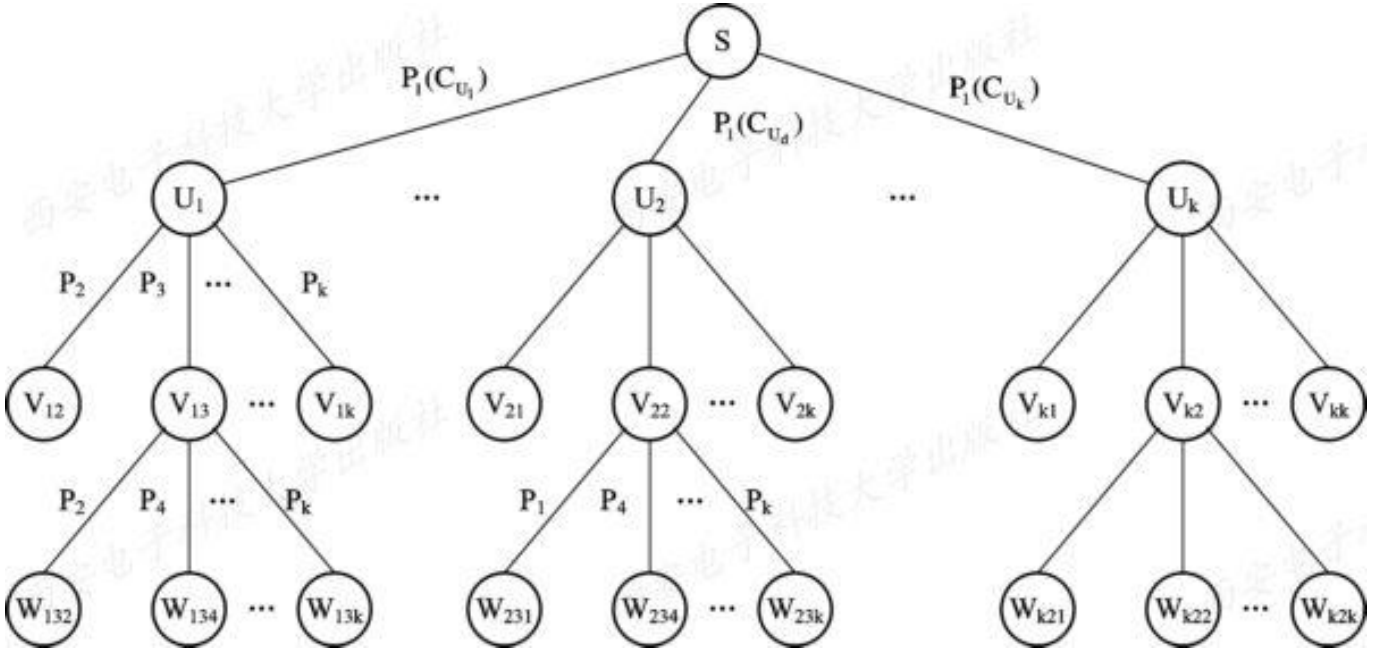


图3-21 付出代价最小的死锁解除算法

P114

30. 在银行家算法的例子中，如果P0发出的请求向量为Request(0, 1, 0)，问系统可否将资源分配给它?

进程 \ 资源情况	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			

作答

31. 在银行家算法中，若出现下表所述资源分配情况，试问：

(1) 该状态是否安全?

(2) 若进程P2提出请求Request(1, 2, 2, 2)后，系统能否将资源分配给它?

Process	Allocation	Need	Available
P_0	0032	0012	1622
P_1	1000	1750	
P_2	1354	2356	
P_3	0332	0652	
P_4	0014	0656	

作答

☆ Consider a system consisting of *m* resources of the same type being shared by *n* processes. A process can request or release only one resource at a time. Show that the system is deadlock free if the following two conditions hold:

a. The maximum need of each process is between 1 and *m* resources

b. The sum of all maximum needs is less than *m* + *n*

Using the terminology of Section Section 7.6.2, we have:

a. $\Sigma_i Max_i < m + n$

b. $Max_i \geq 1$ for all i

Proof: $Need_i = Max_i - Allocation_i$

If there exists a deadlock state then:

c. $\Sigma_i Allocation_i = m$

Use a. to get: $\Sigma Need_i + \Sigma Allocation_i = \Sigma Max_i < m + n$

Use c. to get: $\Sigma Need_i + m < m + n$

Rewrite to get: $\Sigma_i Need_i < n$

This implies that there exists a process P_i such that $Need_i = 0$. Since $Max_i \geq 1$ it follows that P_i has at least one resource that it can release.

Hence the system cannot be in a deadlock state.

主观题 10分

设置

HUAZHONG AGRICULTURAL UNIVERSITY

在本课堂上，你想学习那些内容？
你有那些建议？

作答

正常使用主观题需2.0以上版本雨课堂

投票(匿名) 最多可选5项

设置

HUAZHONG AGRICULTURAL UNIVERSITY

在线听课评价:

A 教师准备充分,讲述条理清晰

B 能调动学习的积极性

C 教学方式单一、缺乏互动

D 学生的收获少，获得感不强

E 内容枯燥、听不懂

提交