海南大学
HAINAN UNIVERSITY

《操作系统原理及安全》

教师：秦小立

学院：网络空间安全学院

邮箱：xlqin@hainanu.edu.cn

办公地点：学院309

课程知识导图

第1章	操作系统引论
第2章	**进程的描述与控制**
第3章	**处理机调度与死锁**
第4章	**进程同步**
第5章	**存储器管理**
第6章	**虚拟存储器**

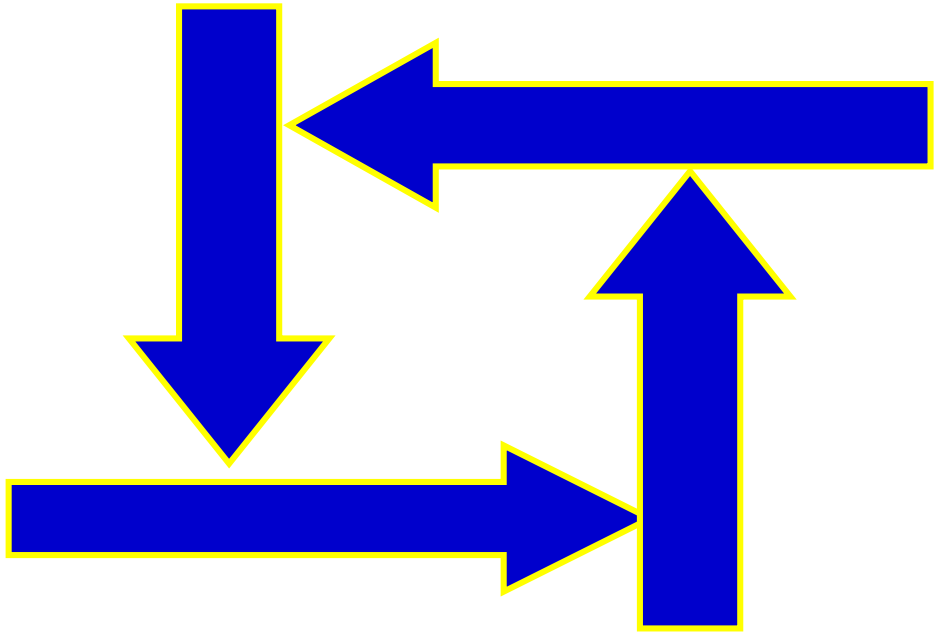
第7章	**输入/输出系统**
第8章	文件管理
第9章	**磁盘存储器管理**
第10章	多处理机操作系统
第11章	虚拟化和云计算
第12章	保护和安全


处理机调度与死锁
处理机调度
处理机调度概述
调度算法
先来先服务调度算法
短作业优先调度算法
优先级调度算法
轮转调度算法
多级队列调度算法
多级反馈队列调度算法
基于公平原则的调度算法
实时调度
调度实例
死锁
死锁概述
预防死锁
避免死锁
银行家算法
死锁的检测与解除

第3章 处理机调度与死锁

3.5 死锁概述

死锁（Deadlock）：指多个进程在运行过程中因争夺资源而造成的一种僵局，当进程处于这种僵持状态时，若无外力作用，这些进程都将永远不能再向前推进。



3.5.1 资源问题

◆1、可重用资源和消耗性资源

可重用性资源是指可重复的使用资源，如打印机。

性质:

- (1)每一类可重用性资源的单元数目相对固定，进程运行期间既不创建也不删除;
- (2) 其每一个单元只能分配给一-个进程、不能共享;
- (3) 进程使用可重用性资源须按顺序:
- a、请求资源; b、使用资源; c、释放资源

3.5.1 资源问题

◆1、可重用资源和消耗性资源

可消耗性(临时性)资源：是指由一个进程产生，被另一进程使用后就再也无用的资源，也称为临时性资源。

性质:

- (1)每一类资源的单元数目在进程运行期间可以不断变化;
- (2) 可由进程不断创造，以增加某类资源的数量;
- (3)进程请求一定数量临时资源用完即弃。

3.5.1 资源问题

◆2、可抢占性资源和不可抢占性资源

（1）可抢占性(可剥夺性)资源：是指进程在获得这类资源后，该资源可再被其他进程或系统抢占，如处理机、内存等。这类资源是不会引起死锁的。

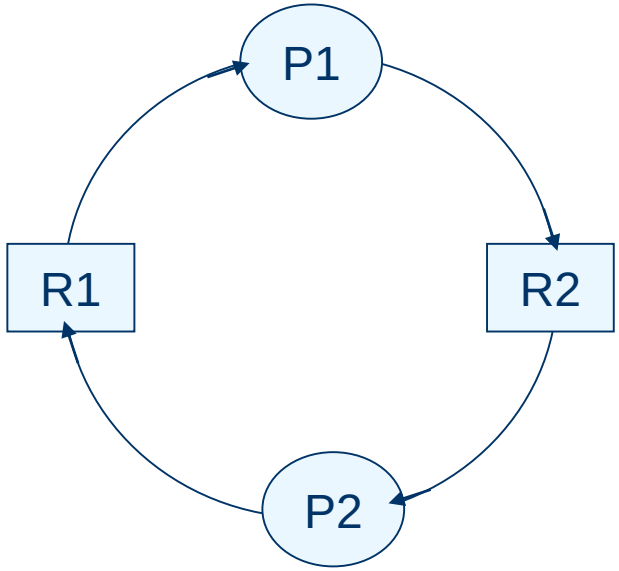
（2）不可抢占性(非剥夺性)资源：是指当系统把这类资源分配给某个进程后，再不能强行收回，只能在进程用完后自行释放，如刻录机、磁带机、打印机等。

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

- 系统中的**不可抢占性资源**，由于它们的数量不能满足诸进程运行的需要，会使进程在运行过程中，因争夺这些资源而陷入僵局。

如，打印机R1,磁带机R2，可供进程P1和P2共享，假定P1已占用R1，P2已占用R2，此时P2又要求R1，因得不到进入阻塞状态，P1又要求R2，也得不到而进入阻塞状态，产生死锁。



3.5.2 计算机系统中的死锁

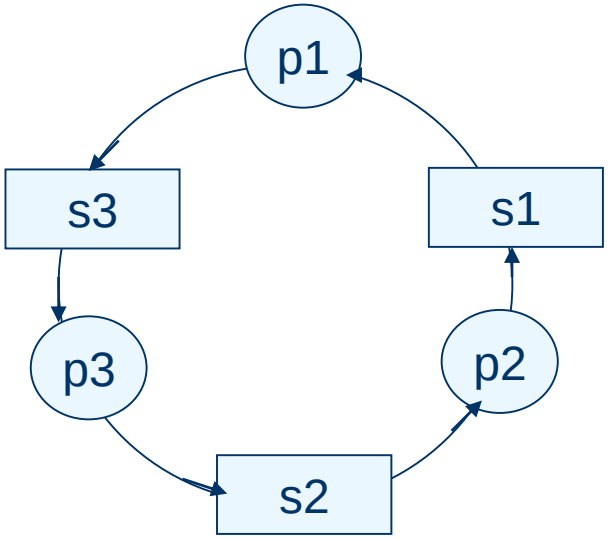
◆2、竞争可消耗性资源引起死锁

- 打印机之类的资源属于可顺序重复使用型资源，称为永久性资源。与之相对的是临时性资源，是指由一个进程产生，被另一进程使用一短暂时间后便无用的资源，也称为消耗性资源，它也可能引起死锁。

P1: ...Request(s3); release(s1)…

P2: ...Request(s1); release(s2)…

P3: ...Request(s2); release(s3)…



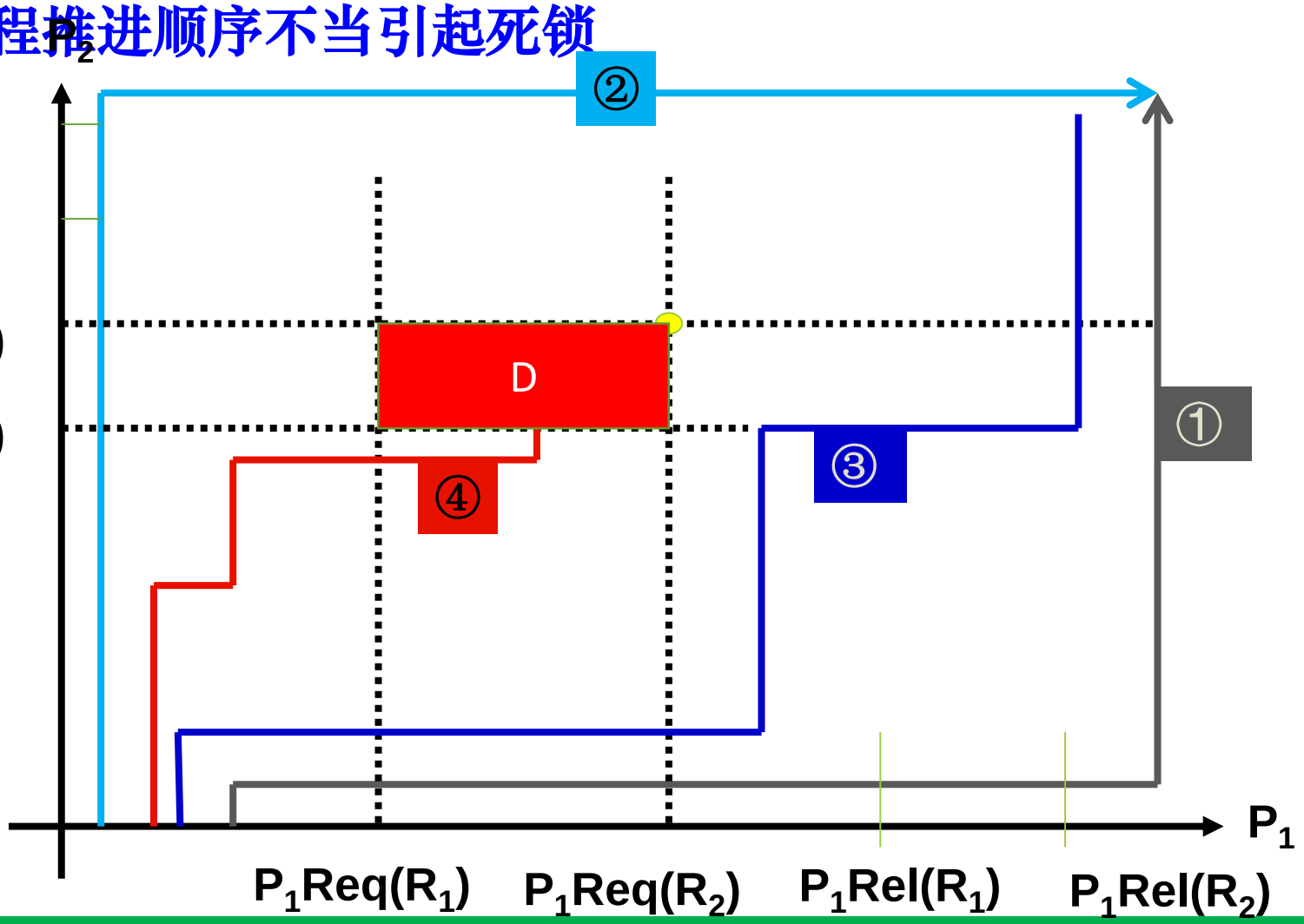
3.5.2 计算机系统中的死锁

◆3、进程推进顺序不当引起死锁

- 由于进程在运行中具有异步性特征，这就可能使上述P1和P2两个进程按下述两种顺序向前推进。
- 进程推进顺序合法
- 进程推进顺序非法

3.5.2 计算机系统中的死锁

◆ 3、进程推进顺序不当引起死锁



3.5.3 死锁定义、必要条件与处理方法

◆1、死锁的定义

死锁：如果一组进程中的每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发生，那么该组进程是死锁。

- **死锁的理解：**参与死锁的进程至少是两个；

  参与死锁的进程至少有两个已经占有资源；

  参与死锁的进程都在等待资源；

  参与死锁的进程是当前系统中所有进程的子集。

- **死锁的危害：**

  死锁的发生将会浪费大量系统资源，甚至导致系统崩溃。

3.5.3 死锁定义、必要条件与处理方法

◆2、死锁的必要条件

1）互斥条件

2）请求和保持条件

3）不抢占条件

4）循环等待条件

3.5.3 死锁定义、必要条件与处理方法

◆2、死锁的必要条件

1）互斥条件

进程访问的是临界资源，即在一段时间内某资源只由一个进程占用。如果此时还有其他进程请求该资源，则请求者只能等待，直至占有该资源的进程用完释放。

2）请求和保持条件

一进程在请求新的资源的同时，保持对已分配资源的占有。

3.5.3 死锁定义、必要条件与处理方法

◆2、死锁的必要条件

3）不抢占条件

指进程已获得的资源，在未使用完之前，不能被抢占，只能在使用完时由自己释放。

4）循环等待条件

指在发生死锁时，必然存在一个进程—资源的环形链。

即进程集合{P0,P1,P2,…,Pn}中的P0正在等待一个P1占用的资源；P1正在等待一个P2占用的资源，……，Pn正在等待一个已被P0占用的资源。

3.5.3 死锁定义、必要条件与处理方法

◆3、死锁处理方法

处理死锁的3种主要策略：

1）确保系统永远不会进入死锁状态

死锁预防和死锁避免

2）允许系统进入死锁状态，然后恢复系统

检测死锁和解除死锁

3）忽略这个问题，假装系统中从未出现过死锁。

这个方法被大部分的操作系统采用，包括UNIX

3.5.3 死锁定义、必要条件与处理方法

◆3、死锁处理方法

在系统中已经出现死锁后，则应及时检测到死锁的发生，并采取适当措施来解除死锁。

目前处理死锁的方法可归结为四种：

1）预防死锁

2）避免死锁

3）检测死锁

4）解除死锁

3.5.3 死锁定义、必要条件与处理方法

◆3、死锁处理方法

1）预防死锁

事先预防，通过设置某些限制条件，去破坏产生死锁的四个必要条件的一个或几个。

2）避免死锁

事先预防，在资源动态分配过程中，不破坏产生死锁的必要条件，而是用某种方法去防止系统进入不安全状态。

3）检测死锁

允许死锁，检测死锁的发生，确定相关进程与资源消除已经发生的死锁。

4）解除死锁

检测到系统中已经发生死锁时，采取相应措施，将进程从死锁状态中解脱出来。

3.5.3 死锁定义、必要条件与处理方法

◆3、死锁处理方法

1）预防死锁

方法：是一种较简单和直观的事先预防方法。该方法是通过设置某些限制条件，去破坏产生死锁的四个必要条件的一个或几个，来预防发生死锁。

优缺点：

预防死锁是一种较易实现的方法，已被广泛使用。但由于所施加的限制条件往往太严格，可能会导致系统资源利用率和系统吞吐量降低。

3.5.3 死锁定义、必要条件与处理方法

◆3、死锁处理方法

2）避免死锁

方法：该方法同样是属于事先预防的策略，这种方法不是预先加上各种限制条件以预防产生死锁的可能性，而是用某种方法去防止系统进入不安全状态，使死锁不致于最终发生。

➢ 优缺点：

这种方法只须事先加以较弱的限制条件，便可获得较高的资源利用率及系统吞吐量，但在实现上有一定的难度。

目前在较完善的系统中，常用此方法来避免发生死锁。

3.5.3 死锁定义、必要条件与处理方法

◆3、死锁处理方法

3）检测死锁

方法：这种方法并不须事先采取任何限制性措施，也不必检查系统是否已经进入不安全区。

优缺点：

此方法允许系统在运行过程中发生死锁，但可通过系统所设置的检测机构，及时的检测出死锁的发生，并精确的确定与死锁有关的进程和资源；然后采取适当的措施，从系统中将已发生的死锁清除掉。

3.5.3 死锁定义、必要条件与处理方法

◆3、死锁处理方法

4）解除死锁

方法：是与死锁检测相配套的一种措施。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。

常用的实施方法是撤销或挂起一些进程，以便回收一些资源，再将这些资源分配给已处于阻塞状态的进程，使之转为就绪状态，以继续运行。

优缺点：死锁的检测与解除措施，有可能使系统获得较好的资源利用率和吞吐量，但在实现上难度也最大。

3.5.4 资源分配图

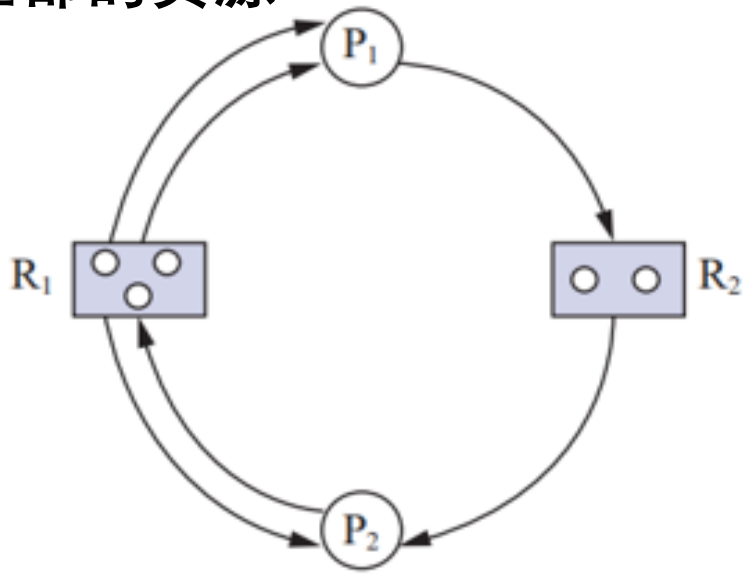
◆资源分配图

系统死锁可利用资源分配图来描述。一个顶点的集合V和边的集合E。V被分为两个部分：

- P = {P_1, P_2, …, P_n}, 含有系统中全部的进程
- R = {R_1, R_2, …, R_m}, 含有系统中全部的资源

请求边：有向边$P_i \rightarrow R_j$

分配边：有向边 $R_j \rightarrow P_i$



第3章 处理机调度与死锁

3.6 死锁预防

◆预防死锁基本原理

➢ **方法思想：**通过设置某些限制条件，去破坏死锁四个必要条件中的一个或多个，来防止死锁。

1. 互斥条件—互斥条件是共享资源必须的，不仅不能改变，还应加以保证
2. 请求和保持条件
3. 不抢占条件
4. 循环等待条件

可以破坏

3.6.1 破坏“请求和保持”条件

◆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。该保证可通过以下两个不同的协议实现

全部分配
全部释放

第一种协议：所有进程在开始运行之前，必须一次性地申请其在整个运行过程中所需的全部资源。该进程在整个运行期间便不会再提出资源要求，从而破坏了“请求”条件。

分配资源时只要进程请求的资源有一种不能满足，就不给该进程分配任何资源，即该进程等待时不占有任何资源，从而破坏了“保持”条件。

3.6.1 破坏“请求和保持”条件

◆破坏“请求和保持”条件

优点：简单、易于实现且安全。

缺点：

① 资源浪费，严重恶化了资源利用率。进程一次性获得所需全部资源，其中有些资源很少或很晚才使用。

②经常发生饥饿现象。用户作业必须等待，直到所有资源满足才能运行，所以就可能出现迟迟得不到全部资源。

3.6.1 破坏“请求和保持”条件

◆破坏“请求和保持”条件

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，等到已经获得的资源使用完毕释放后，再去申请后期运行需要的资源。

第二种协议要比第一种好。减少饥饿出现的可能。

3.6.2破坏“不可抢占”条件

◆破坏“不可抢占”条件

含义：进程占有的资源只有等该进程用完后由其自行释放。

如何破坏?

- 一个已拥有某些资源的进程，若它再提出新资源要求而不能立即得到满足时，它必须主动释放其已拥有的全部资源，以后需要时再重新申请。这破坏了“不可抢占”条件。此处进程只要拥有部分资源就运行，是一种动态资源分配。

缺点：该方法实现比较复杂，代价较大。还会延长进程的周转时间，增加系统开销，降低系统吞吐量。

3.6.3 破坏“循环等待”条件

◆破坏“循环等待”条件

如何破坏?

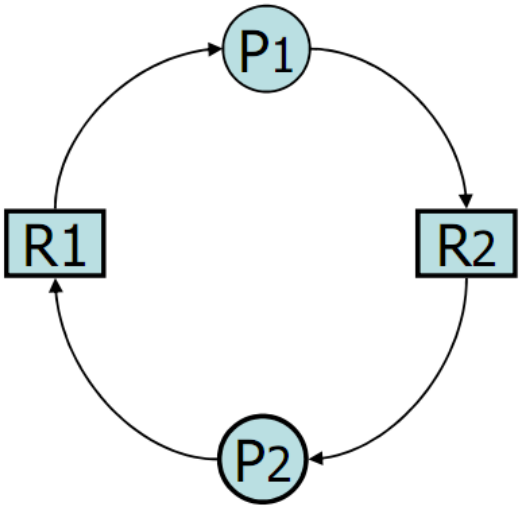
- 对所有的资源类型进行线性排序，并赋予不同的序号，要求进程按照递增顺序申请资源。
- 优点：同前两种方法相比,其资源利用率和系统吞吐量有较明显的改善。
- 缺点： (1)限制了新类型设备的增加； (2)作业使用资源的顺序与系统规定的顺序不同，造成对资源的浪费； (3)限制了用户简单、自主的编程。

3.6.3 破坏“循环等待”条件

◆破坏“循环等待”条件

思考题：

为什么对所有资源进行线性排序,按照资源序号递增的次序分配，就可以解决死锁问题呢？证明该方法可以预防死锁的发生。



证明:采用反证法，假设采用破坏循环等待条件依旧出现死锁。

由图可知，P1进程占有R1申请R2则说明R1的序号小于R2的序号;

同理，P2进程占有R2申请R1 ,说明R2的序号小于R1的序号，这是一个矛盾的结果，所以，假设不成立，证毕。

3.6 预防死锁-小结

◆破坏“循环等待”条件

预防死锁通过设置某些限制条件以破坏死锁四个必要条件中的一个或多个，来防止死锁。

- 互斥条件一般不允许破坏。
- 破坏请求和保持条件的资源分配算法是一种静态资源分配。
- 破坏不可抢占条件和循环等待条件是动态资源分配算法。
- 破坏循环等待条件是相对优秀的一种预防死锁算法。

第3章 处理机调度与死锁

3.7 死锁避免

◆死锁避免

死锁避免仍然是预防策略，是在资源分配过程中防止进入不安全状态。

- 设一个简单而有效的模型，要求每一个进程声明它所需要的资源的最大数。
- 死锁避免算法动态检查资源分配状态以确保不会出现循环等待的情况。
- 资源分配状态定义为可用的与已分配的资源数，和进程所需的最大资源量。

3.7.1 系统安全状态(避免死锁)

◆1、系统安全状态

基本思想：在系统运行过程中，对进程提出的每一个(系统能够满足的)资源申请进行安全性检查，如果安全，则予以分配。否则不予分配。

- 安全状态是指系统能按某种进程顺序(Pm, Pk,…,Pn)(该序列为安全序列)，为每个进程分配其所需资源，直至满足每个进程对资源的最大需求，使每个进程都可顺利地完成。
- 如果系统无法找到一个安全序列，则称系统处于不安全状态。

3.7.1 系统安全状态(避免死锁)

◆2、安全状态实例

有进程 P1，P2，P3，已知系统共有 12 台资源，当前各进程的资源需求和资源分配情况如下：

进程	最大需求	已分配	还需	可用资源
P1	10	5	5	3
P2	4	2	2	
P3	9	2	7	

问题：1、设此时为T0时刻，该时刻是否安全?

解：在T0时刻存在安全序列<P2,P1,P3>，该时刻是安全的。

3.7.1 系统安全状态(避免死锁)

◆3、由安全状态进入不安全状态

若不按安全序列分配资源，则可能由安全状态进入不安全状态。

例如，上例中，T0 时刻后 P3 进程申请 1 台，分配给它后资源分配情况如下表，那么 此时是否安全?

进程	最大需求	已分配	还需	可用资源
P1	10	5	5	2
P2	4	2	2	
P3	9	3	6	

不安全。当 P3进程申请1台资源时，尽管有可用资源，也不能分配给它，必须让 P3等到P1和P2完成释放，才可分配资源。

3.7.1 系统安全状态(避免死锁)

该方法避免死锁的关键就是确保系统始终处于安全状态。即：

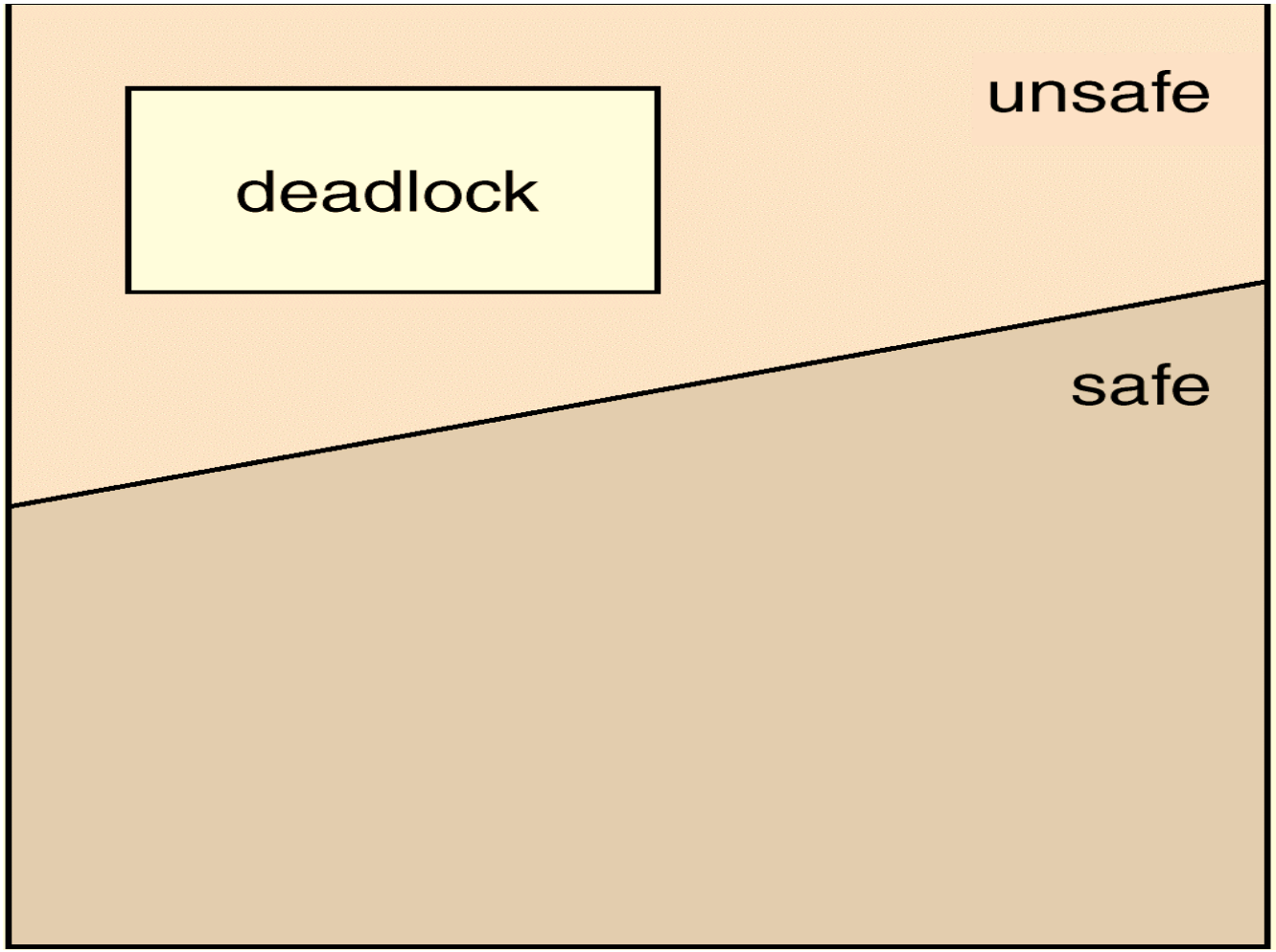
(1)一个系统开始处于安全状态。

(2)当有进程请求一个可用资源时，系统对该进程的请求进行计算，若将资源分配给该进程后系统仍处于安全状态，才将该资源分配给该进程。否则，不能满足其资源请求，令进程等待。

注意：不安全状态等于死锁吗？

- 处于不安全状态的系统不一定会发生死锁(安全性检查中最大资源量是执行前提供的，而进程实际运行时可能小于这个值)
- 处于安全状态的系统一定不会发生死锁

3.7.1 系统安全状态(避免死锁)



3.7.2利用银行家算法避免死锁

银行家算法，（经典的Dijkstra算法）

银行家算法是一个非常经典的避免死锁的算法。

基本思想：当某个进程提出资源申请时，必须先判断把资源分配给该进程后，会不会引起死锁；如果不会，再进行分配。否则，就不分配。

3. 7. 2利用银行家算法避免死锁

1、算法中的数据结构

可用资源向量：Available[j]表示系统现有Rj类可用资源的数量

资源最大需求矩阵(资源总需求矩阵)

Max[i;j]表示进程i对Rj类资源的最大需求数量

资源分配矩阵(已分配资源矩阵).

Allocation[i,j]表示进程i现已共分得Rj类资源的数量

资源需求矩阵

Need[i,j]表示进程i还需要Rj类资源的数量

三个矩阵的关系：Need[i,j]=Max[i,j]— Allocation[i,j]

3.7.2利用银行家算法避免死锁

2、银行家算法

设$Request_i[j]=k$,表示进程Pi需要k个Rj类型的资源

（1）如果$Request_i[j] <= Need[i,j]$,便转向步骤2；否则认为出错，因为它所请求的资源数已超过它所需要的最大值。

（2）如果$Request_i[j] <= Available[j]$,便转向步骤3；

否则，表示尚无足够资源，Pi需等待。

（3）系统试探着把资源分配给进程Pi，并修改下面数据结构中数值 :

$Available[j]:= Available[j]- Request_i[j];$

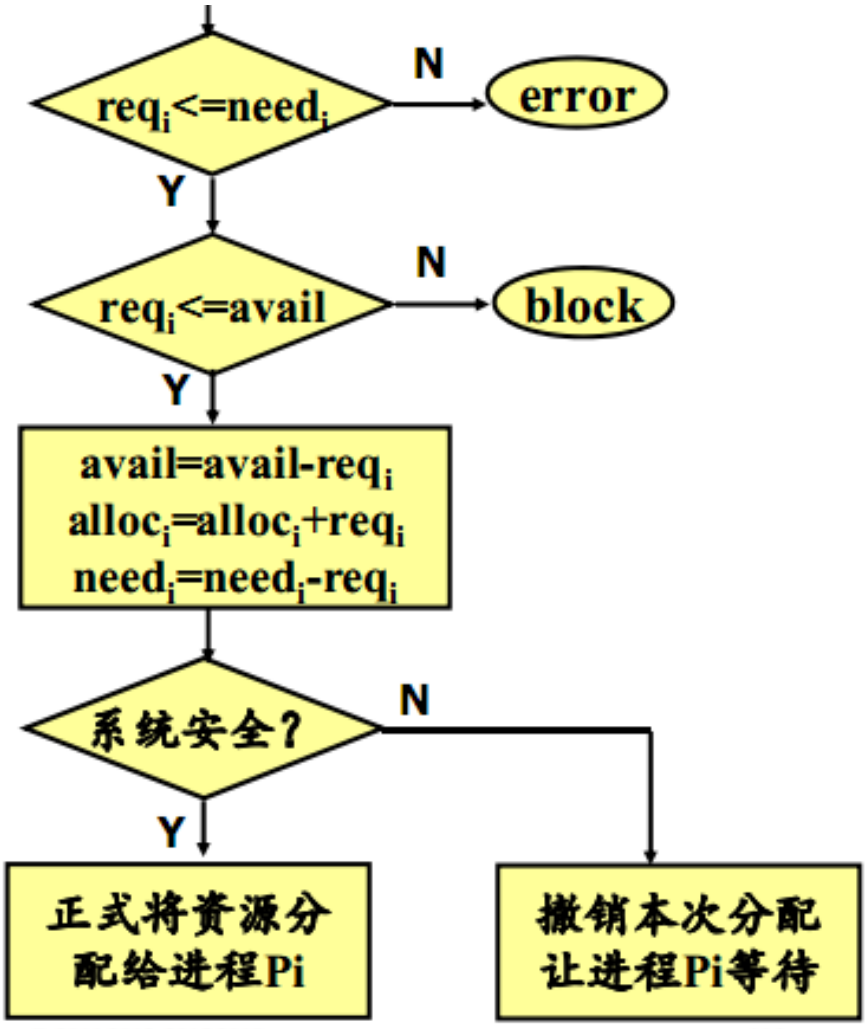
$Allocation[i,j]:=Allocation[i,j]+Request_i[j];$

$Need[i,j]:=Need[i,j]-Request_i[j];$

3.7.2利用银行家算法避免死锁

2、银行家算法

（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以决定是否完成本次分配。若安全，则正式分配资源给进程Pi；否则，不分配资源，让进程Pi等待。



3. 7. 2利用银行家算法避免死锁

3、安全性算法

（1）设置两个向量，并初始化：

工作向量work：表示系统可提供给进程继续运行所需的各类资源数目，含有m个元素的一维数组，初始时，work:=Available;

Finish: 含n个元素的一维数组，表示系统是否有足够的资源分配给n个进程，使之运行完成。开始时先令Finish[i]:=false (i=1..n);

当有足够资源分配给进程i时，再令Finish[i]:=true。

3. 7. 2利用银行家算法避免死锁

3、安全性算法

（2）从进程集合中找到一个能满足下述条件的进程：

Finish[i]=false;Need[i,j]<=work[j]; 若找到，执行步骤（3），否则执行步骤（4）。

（3）当进程P_i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;

goto 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、银行家算法举例

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。

系统中T_0时刻的资源分配情况：

进程 \ 资源情况	Max A	Max B	Max C	Allocation A	Allocation B	Allocation C	Need A	Need B	Need C	Available A	Available B	Available 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			

3.7.2利用银行家算法避免死锁

4、银行家算法举例

问题：(1)考察系统在T0时刻的安全性？（安全性算法检查）

资源情况 / 进程	Work A B C	Need A B C	Allocation A B C	Work+Allocation A B C	Finish
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

3.7.2利用银行家算法避免死锁

4、银行家算法举例

问题:(2)P1请求资源Request1(1,0,2)?

① 银行家算法检查Request ≤ Available，即(1,0,2) ≤(3,3,2)为真，可预分配。

	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			

3.7.2利用银行家算法避免死锁

4、银行家算法举例

问题:(2)P1请求资源Request1(1,0,2)?

② 安全性检查过，正式分配。有安全序列<P_1, P_3, P_4, P_0, P_2>

资源 情况 进程	Work A B C	Need A B C	Allocation A B C	Work+Allocation A B C	Finish
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

3.7.2利用银行家算法避免死锁

4、银行家算法举例

问题：(3)P4请求资源Request4(3,3,0)？

银行家算法：

Requset4 (3,3,0) <=Need4 (4,3,1)

Requset4(3,3,0)>Available(2,3,0)

此时P4等待。

	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	

3.7.2利用银行家算法避免死锁

4、银行家算法举例

问题：（4）P0请求资源Request0(0,2,0)？

① 银行家算法： Requset0(0,2,0)<=Need0 (7,4,3)

Requset0(0,2,0)<=Available(2,3,0)

② 试探分配，修改相关值

Avaliable= (2,3,0)-(0,2,0)=(2,1,0)

Allocation0=(0,1,0)+(0,2,0)=(0,3,0)

Need0=(7,4,3)-(0,2,0)=(7,2,3)

③ 再利用安全性算法检查此时系统是否安全。很容易得到，可用资源Available(2,1,0)已无法满足任一进程的需求，试探作废，不分给P0资源，P0等待。

3.7.2利用银行家算法避免死锁

4、银行家算法举例

问题：（4）P0请求资源Request0(0,2,0)？

	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			

3.7.2利用银行家算法避免死锁

银行家算法总结

(1) 首先判断当前时刻的安全性；(安全性算法)

(2) 若安全，当某进程提出资源申请 Request时，尝试进行预分配(两个比较，并修改相关值)；

(3) 判断预分配后的安全性；(安全性算法)

(4) 若仍安全，则实施预分配；否则，该进程等待。

第3章 处理机调度与死锁

3.8.1 死锁的检测

◆死锁的检测

1）基本原理

- 允许死锁发生，OS不断监视系统进展情况,判断死锁是否发生。
- 一旦死锁发生则采取专门的措施，解除死锁并以最小的代价恢复操作系统运行。

2）死锁检测时机

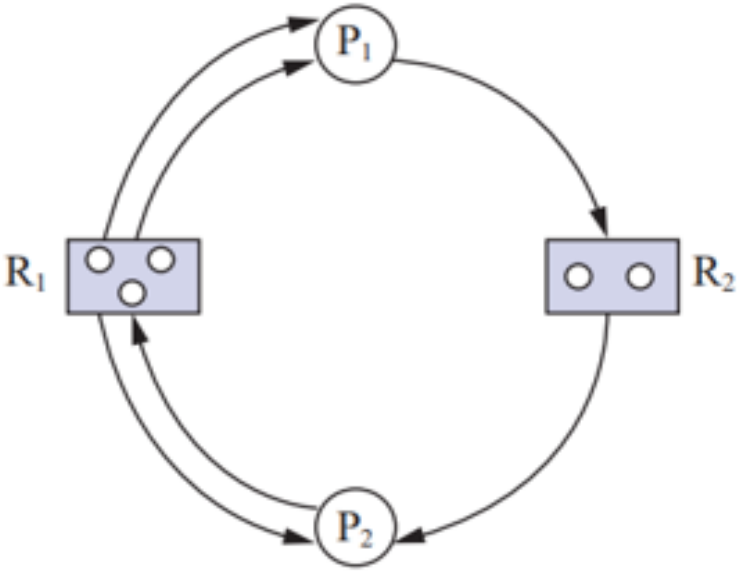
- 时机一:定时检测
- 时机二:当进程阻塞时检测死锁(其缺点是系统的开销大)
- 时机三:系统资源利用率下降时检测死锁

3.8.1 死锁的检测

◆资源分配图

系统死锁可利用资源分配图来描述。

该图由表示进程的圆圈和表示一类资源的方框组成，其中方框中的一个点代表一个该类资源，请求边是由进程指向方框中的R_j，而分配边则应始于方框中的一个点。



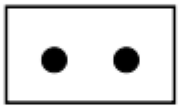
3.8.1 死锁的检测

◆1、死锁定理-资源分配图的简化

➢ 资源分配图的简化

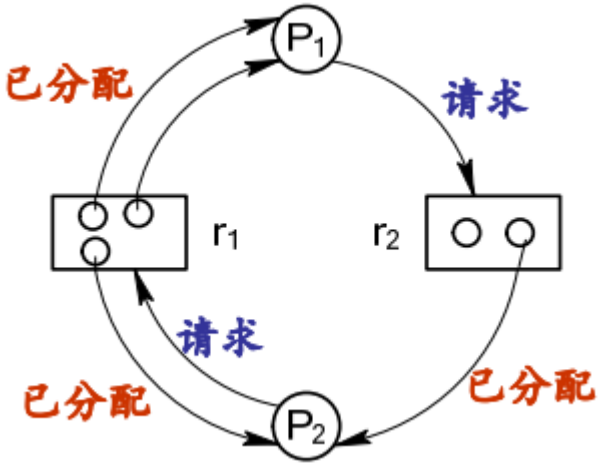
资源结点

- 资源类用方框表示
- 资源实例用方框中的黑圆点（圈）表示



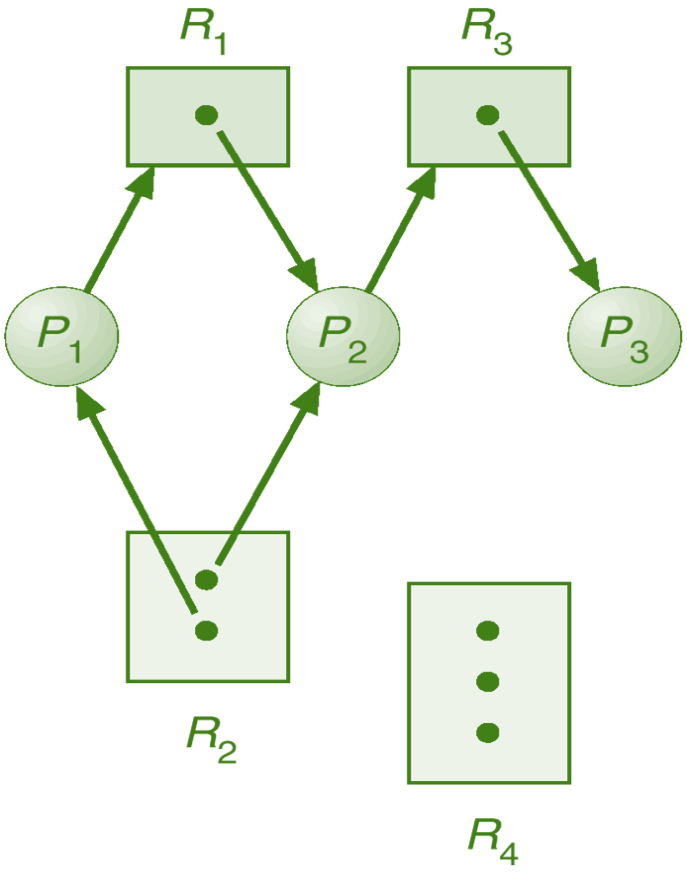
进程结点

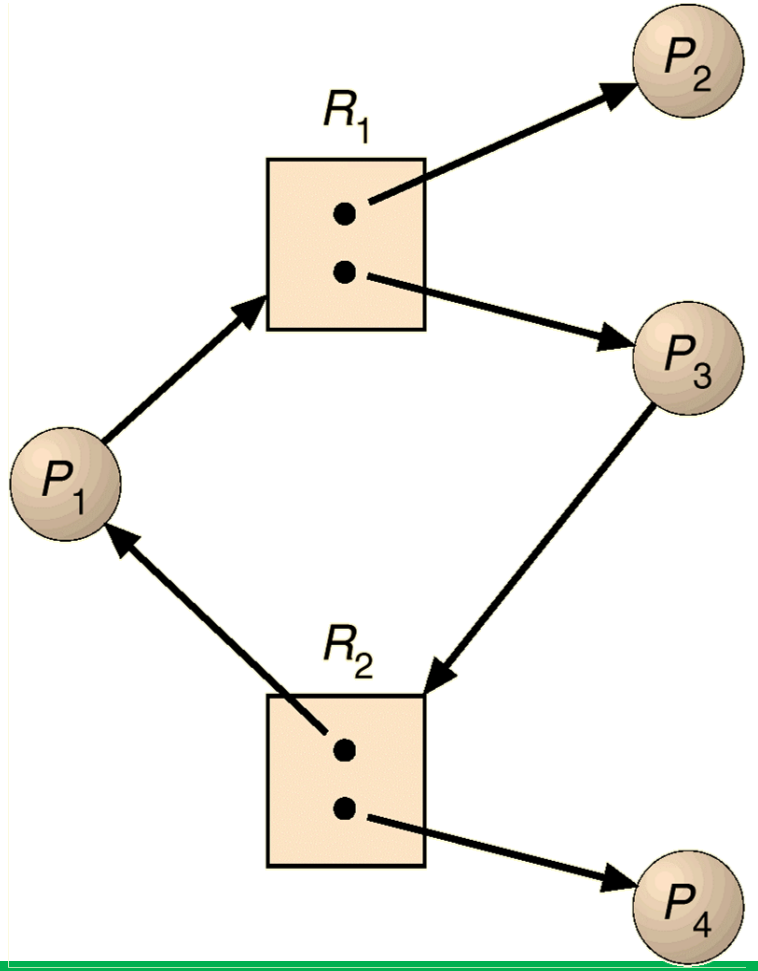
- 用含有进程名的圆圈表示



3.8.1 死锁的检测

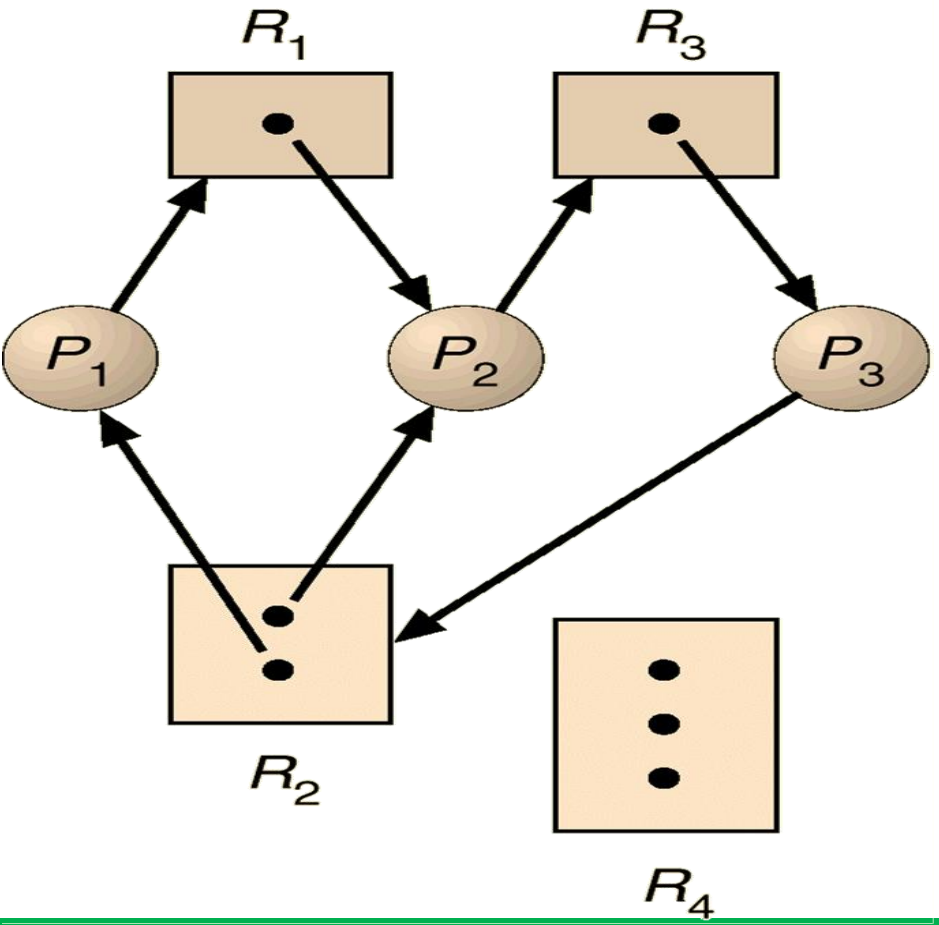
◆无死锁的资源图

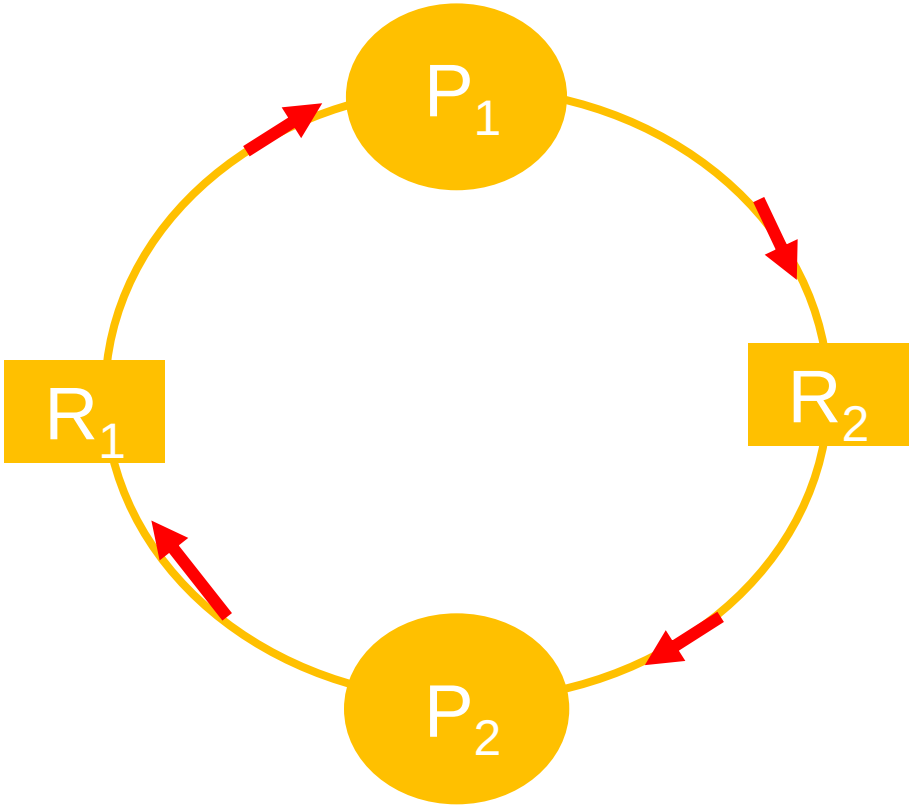




3.8.1 死锁的检测

◆有死锁的资源分配图





3.8.1 死锁的检测

◆基本事实

如果图没有环，那么不会有死锁！

如果图有环，那么：

- 如果每一种资源类型只有一个实例，那么死锁发生；
- 如果一种资源类型有多个实例，那么可能死锁。

3.8.1 死锁的检测

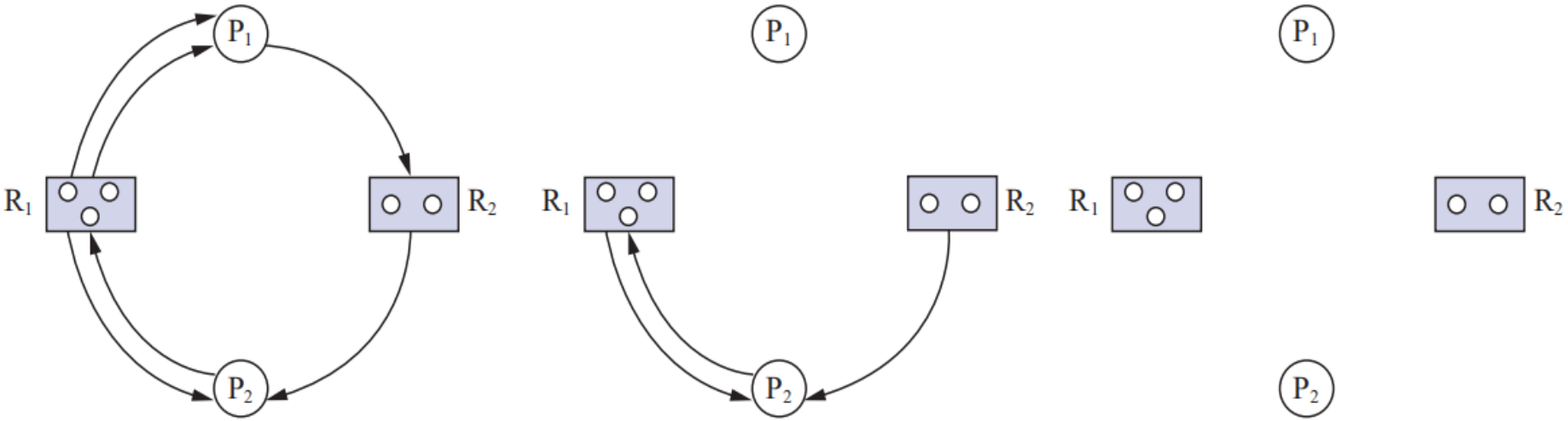
◆1、死锁定理-资源分配图的简化

- 资源分配图化简(进程结点孤立化)
- 1）从分配图中找一个非孤立、非阻塞的进程结点(其所有请求边可转化为分配边，转化后该结点只有分配边)；
- 2）再把相应的资源分配给等待该资源的进程,将这些进程中的相应请求边变为分配边，去掉其请求边和分配边，将其变为孤立结点;
- 3）重复以上步骤，若所有进程都可成为孤立结点，称该图是可完全简化的，否则称该图是不可完全简化的。

3.8.1 死锁的检测

◆1、死锁定理-资源分配图的简化

- 资源分配图的简化



3.8.1 死锁的检测

◆1、死锁定理

- 对于较复杂的资源分配图，可能有多个既未阻塞、又非孤立的进程结点，不同的简化顺序，是否会得到不同的简化图？已经证明：所有的简化顺序，都将得到相同的不可简化图。
- S为死锁状态的充分条件：当且仅当S状态的资源分配图是不可完全简化的。该充分条件称为死锁定理。
- 死锁的必要条件:如果资源分配图中没有环路,则系统中没有死锁，如果图中存在环路，则系统中可能存在死锁。
- 死锁的充要条件:如果每个资源类中只包含一个资源实例，则环路是死锁存在的充分必要条件。

3.8.1 死锁的检测

◆2、死锁检测的数据结构-类似银行家算法（基于资源分配图简化）

Work：=Available；

L：={L_i|$Allocation_i$=0∩$Request_i$=0}/*存放孤立进程点*/

for all $L_i \notin L$ do

begin

for all $Request_i \leq Work$ do

begin

Work：= Work + $Allocation_i$；

L_i∩L；

end

end

deadlock:=┐（L={P_1，P_2，…，P_n}）；/*Deadlock为true时死锁*/

3.8.1 死锁的检测

◆2、死锁检测的数据结构

1）让Work和Finish作为长度为m和n的向量初始化

(a) Work := Available

(b) For i = 1,2, …, n, if $Allocation_i \neq 0$, then

Finish[i] := false;otherwise, Finish[i] := true.

2）找到满足下列条件的下标i

(a) Finish[i] = false

(b) $Request_i \leq Work$

如果没有这样的i存在，转4

3）Work := Work + $Allocation_i$

Finish[i] := true

转 2.

4）如果有一些i, $1 \leq i \leq n$，Finish[i] = false, 则系统处在死锁状态。而且， 如果 Finish[i] = false, 则进程 P_i 是死锁的。

3.8.1 死锁的检测

◆2、死锁检测的数据结构-例子

五个进程P_n到P_4,三个资源类型A（7个实例），B（2个实例），C（6个实例）。时刻T_0的状态：

	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			

对所有*i*，序列 <P_0, P_2, P_3, P_1, P_4> 将导致*Finish*[*i*] = true，因此死锁不存在。

3.8.1 死锁的检测

◆2、死锁检测的数据结构

P2请求一个额外的C实例

	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

系统的状态?

- 可以归还P_0所有的资源，但是资源不够完成其他进程的请求。
- 死锁存在，包括进程P1、P2、P3和P4。

3.8.2 死锁的解除

◆解除死锁的2个方法

1）抢占资源。从一个或多个进程中抢占足够数量的资源给死锁进程，以解除死锁状态。

2）终止或撤消死锁进程。终止系统中一个或多个死锁进程，直到打破循环环路，使死锁状态消除为止。

➢① 终止所有死锁进程（最简单方法）：以前工作全部作废，损失可能很大

➢② 逐个终止进程（稍温和方法）：应使撤消进程所付出的代价尽可能小，选取被撤消进程的参考因素：

3.8.2 死锁的解除

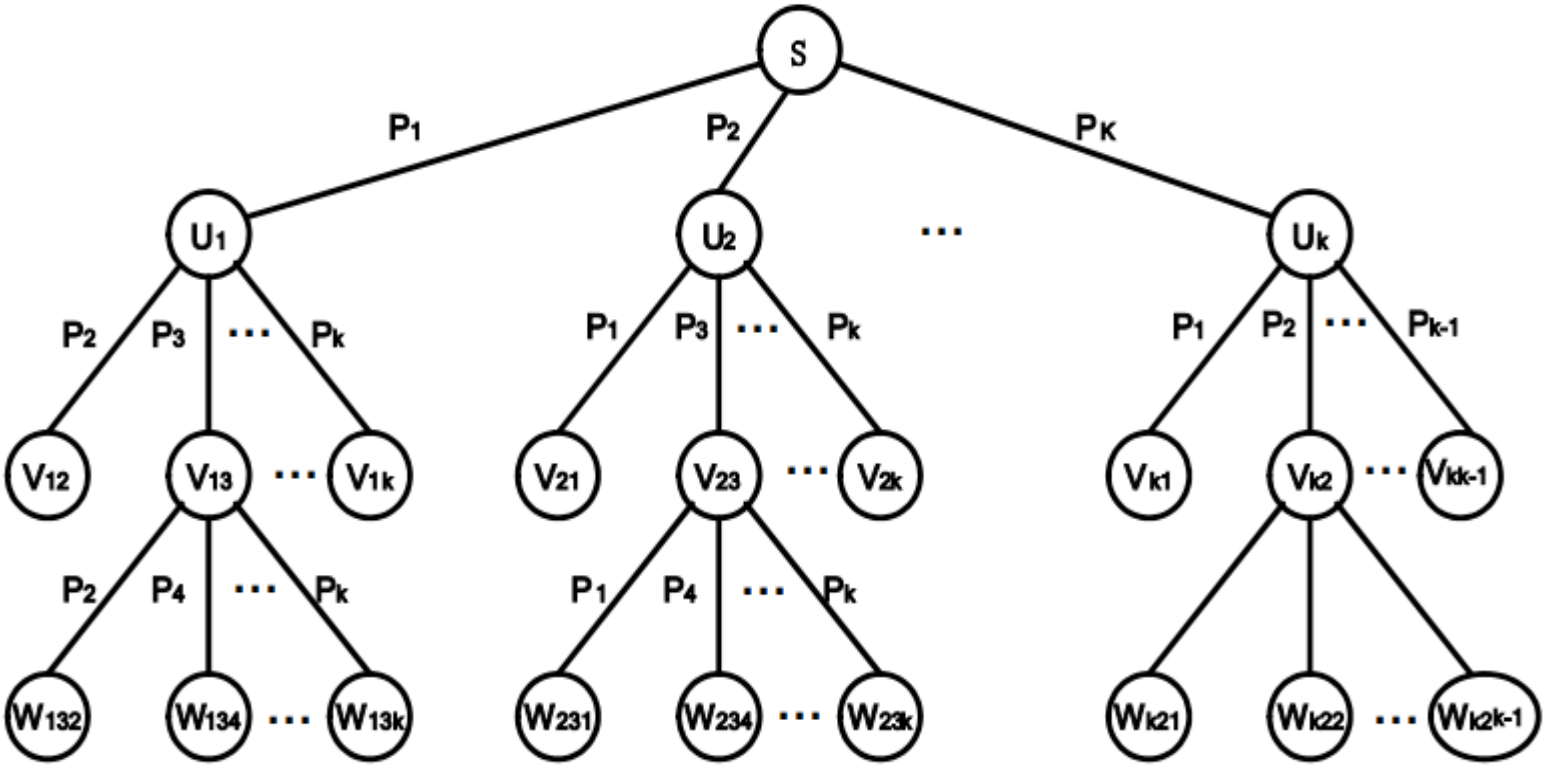
◆1、终止死锁进程的方法

② 终止或撤消进程的方法

- 进程的优先级大小；
- 进程已执行多长时间，还要多长时间
- 进程已使用多少资源，还需多少资源；
- 进程是交互式的还是批处理式。

3.8.2 死锁的解除

◆2、付出代价最小的死锁解除算法:



撤消策略：1）按树的广度优先搜索方法撤消：代价可能太大

2）按树的最短路径(最小代价)优先方法撤消：比较好

3.8.2 死锁的解除

◆补充：用经验公式判断死锁

问题1:设系统仅有一类数量为M的独占型资源,系统中N个进程竞争该类资源，其中各进程对该类资源的最大需求均为W。当M, N, W分别取下列值时，试判断下列哪些情况可能会发生死锁?

(1) M=2, N=2, W=2;

(2) M=3, N=2，W=2；

(3) M=3, N=2, W=3；

(4) M=5，N=3, W=2；

(5) M=6，N=3，W=3;

3.8.2 死锁的解除

◆补充：用经验公式判断死锁

死锁判断的经验公式：

$$x=\begin{cases}1 & M \le N \\ 1+\dfrac{M-1}{N} & M > N\end{cases}$$

若W≤x，则不会发生死锁;若W>x ,则可能导致死锁。

3.8.2 死锁的解除

◆补充：用经验公式判断死锁

(1) M=2, N=2, W=2;	x=1,x<W，可能会死锁；
(2) M=3, N=2，W=2；	x=2， x=W不会死锁；
(3) M=3, N=2, W=3；	x=2，x<W可能会死锁；
(4) M=5，N=3, W=2；	x=7/3，x>W,不会死锁；
(5) M=6，N=3，W=3;	x= 8/3，x<W,可能会死锁。

3.8.2 死锁的解除

◆补充：用经验公式判断死锁

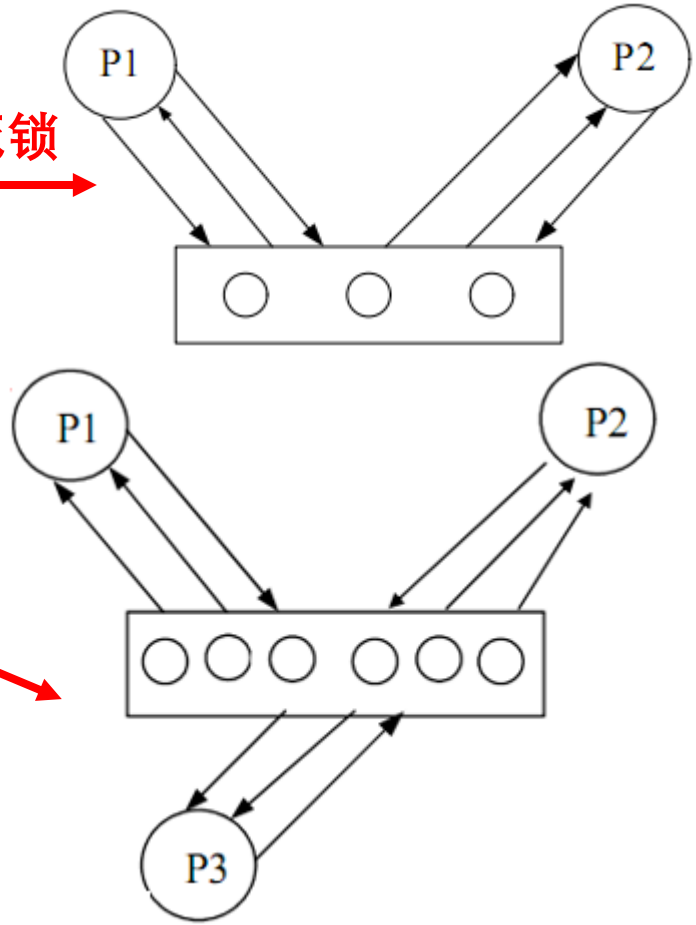
(1) M=2, N=2, W=2;

(2) M=3, N=2，W=2；

(3) M=3, N=2, W=3；

(4) M=5，N=3, W=2；

(5) M=6，N=3，W=3;



3.8.2 死锁的解除

◆补充：用经验公式判断死锁

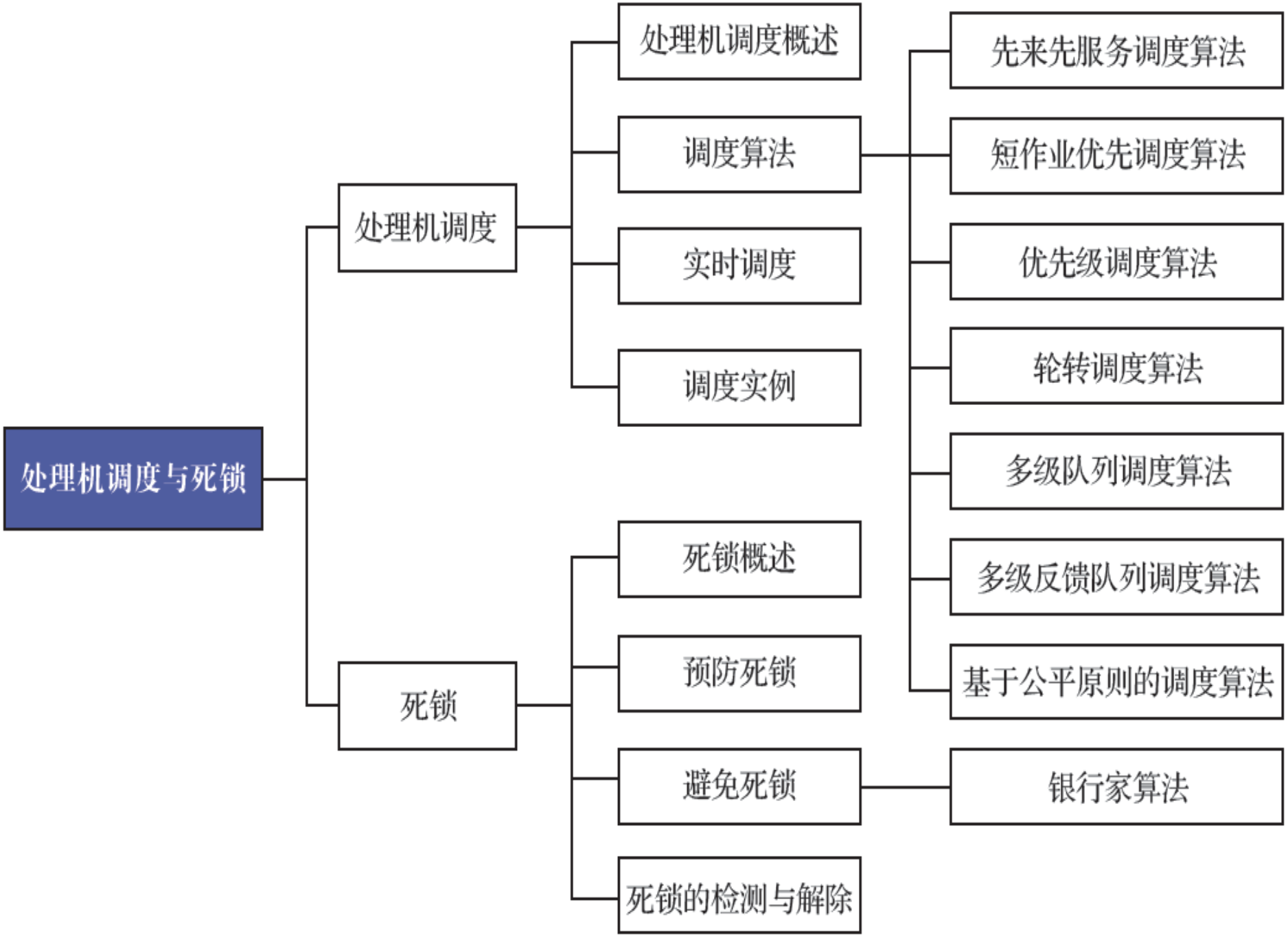
问题2:一台计算机有10台磁带机被n个进程竞争每个进程最多需要3台磁带机，那么n最多为？？时，系统没有死锁的危险?

解:依题意,W= 3,M=10，x=9/n+1

由经验公式如要不死锁,则需x>=W

即，9/n+1>= 3n<=4.5

故nmax=4


处理机调度与死锁
处理机调度
处理机调度概述
调度算法
先来先服务调度算法
短作业优先调度算法
优先级调度算法
轮转调度算法
多级队列调度算法
多级反馈队列调度算法
基于公平原则的调度算法
实时调度
调度实例
死锁
死锁概述
预防死锁
避免死锁
银行家算法
死锁的检测与解除

第3章 处理机调度与死锁

本章-总结

- 处理机调度的三个层次**
- 进程调度的两种方式**
- 处理机调度算法的目标
- 三种作业调度算法**

  FCFS、SJF、HRRN
- 三种进程调度算法**

  RR、FB、公平
- 两种常用实时调度算法(EDF、LLF)及优先级倒置**
- 产生死锁的三类原因
- 死锁的四大必要条件**
- 死锁的预防*

第3章 处理机调度与死锁

本章-小结

- 产生死锁的三类原因
- 死锁的四大必要条件**
- 死锁的预防:*
- 死锁的避免(银行家算法)**.
- 死锁的检测**

死锁定理、资源分配图化简、死锁检测算法死锁的解除

- 重要概念:

作业、作业步、作业流、非抢占方式、抢占方式、周转时间、带权周转时间、静态优先权、动态优先权、响应比、松驰度、安全状态、死锁定理、动态分配、静态分配

第3章 处理机调度与死锁

作业

课后习题

- 需写作业本：
  - 简答：1,6,10,15,
  - 计算：20,22
  - 综合：25

海南大学
HAINAN UNIVERSITY

本章结束，
祝同学们学习愉快！