7 死锁的预防、避免与检测、银行家算法（书 3.7 p126-137）

一、结合 PPT、考点说明与参考资料进行以下梳理

✧ 背景：多个进程并发执行可以有效提高系统资源利用率和系统吞吐量，但当多个进程竞争系统**有限且不可抢占的**资源时，可能产生死锁。

✧ 整体脉络：

死锁的定义 必要条件和处理方法

（一）死锁的定义

如果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（Deadlock）。

（二）产生死锁的必要条件

虽然进程在运行过程中可能会发生死锁，但产生死锁是必须具备一定条件的。产生死锁必须同时具备下面四个必要条件，只要其中任意一个条件不成立，死锁就不会产生：

（1）互斥条件。进程对所分配到的资源进行排他性使用，即在一段时间内，某资源只能被一个进程占用。如果此时还有其他进程请求该资源，则请求进程只能等待，直至占有该资源的进程用毕释放。

（2）请求和保持条件。进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源已被其他进程占有，此时请求进程被阻塞，但对自己以获得的资源保持不放。

（3）不可抢占条件。进程已获得的资源在未使用完之前不能被抢占，只能在进程使用完时由自己释放。

（4）循环等待条件。在发生死锁时，必然存在一个进程—资源的循环链，即进程集合{P0,P1,P2,P3,...,Pn}中的P0正在等待P1占用的资源，P1正在等待P2占用的资源，……，Pn正在等待已被P0占用的资源。

（三）处理死锁的方法

（1）预防死锁。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中的一个或几个来预防产生死锁。

（2）避免死锁。在资源的动态分配过程中，用某种方法防止系统进入不安全状态，从而可以避免产生死锁。

（3）检测死锁。通过检测机构及时地检测出死锁的发生，然后采取适当的措施，把进程从思索中解脱出来。

（4）解除死锁。当检测到系统中已发生死锁时，就采取相应的措施，将进程从死锁状态中解脱出来。常用方法是---撤销一些进程，回收他们的资源，将他们分配给已处于阻塞状态的进程，使其能继续运行。

目标考点：

（1）什么是死锁：

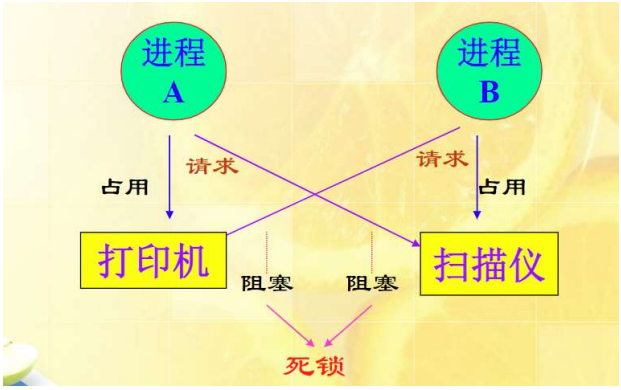
1. 系统中存在一组进程（两个及以上且至少有两个已经占有资源），且它们每个都无限等待被该组进程中另一进程所占用的且永远无法释放的资源，这种现象称为**进程死锁**，这一组进程就称为**死锁进程**。
   注：死锁发生会浪费大量系统资源，甚至导致系统崩溃。

（2）产生死锁的原因及其四个必要条件

Ⅰ产生死锁的原因：

a) 竞争**不可抢占的**资源

①竞争不可剥夺资源（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，如打印机、队列、文件、磁带机）产生死锁



注：系统资源分为可重用资源与**消耗性资源**，其中可重用资源包括可剥夺资源（如处理器 CPU、主存等）、**不可剥夺资源**；

可重用资源（永久性资源）：可被多个进程多次使用，如所有硬件。

- ✧ 只能分配给一个进程使用，不允许多个进程共享。
- ✧ 进程在对可重用资源的使用时，须按照请求资源、使用资源、释放资源这样的顺序。
- ✧ 系统中每一类可重用资源中的单元数目是相对固定的，进程在运行期间，既不能创建，也不能删除。

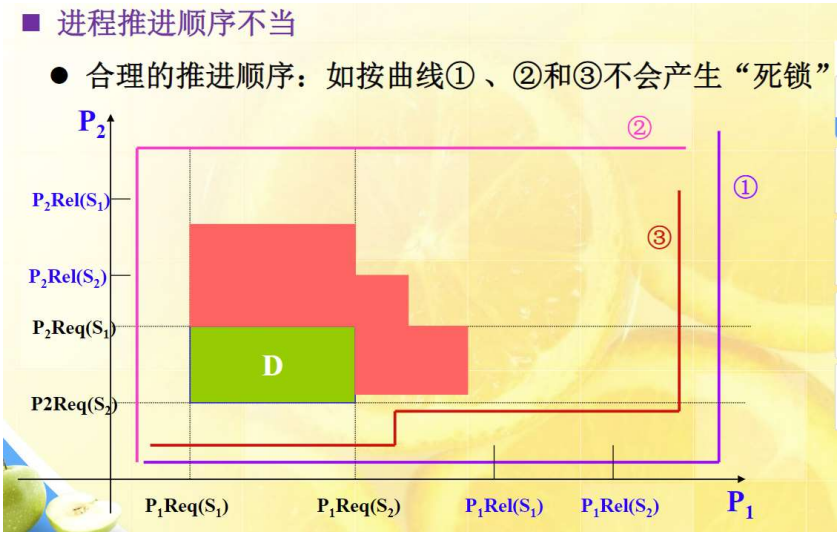
②争夺消耗性资源（由产生，被进程使用、使用后消失的资源，如中断、信号、信息等）产生死锁

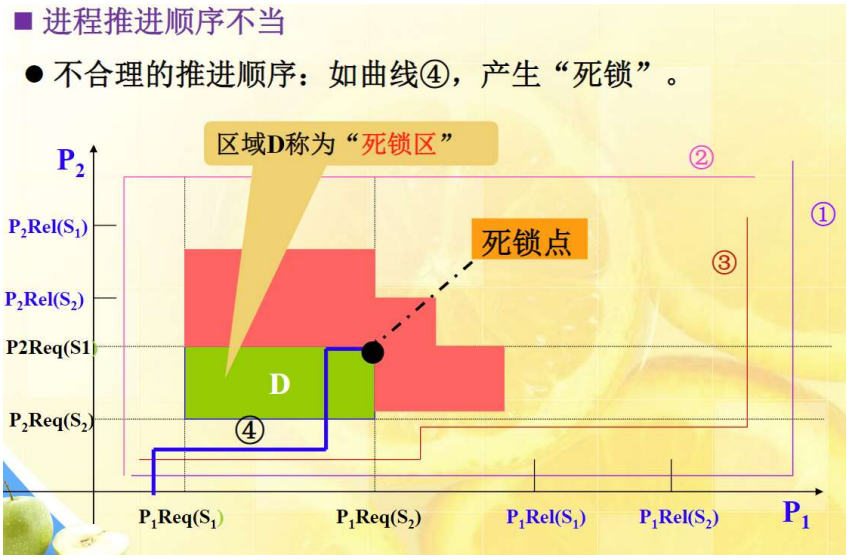
消耗性资源（临时性资源）：又称临时性资源，是由进程在运行期间动态的创建和消耗的。

- ✧ 消耗性资源在进程运行期间是可以不断变化的，有时可能为 0。
- ✧ 进程在运行过程中，可以不断地创造可消耗性资源的单元，将它们放入该资源类的缓冲区中，以增加该资源类的单元数目。
- ✧ 进程在运行过程中，可以请求若干个可消耗性资源单元，用于进程自己消耗，不再将它们返回给该资源类中。
- ✧ 可消耗资源通常是由生产者进程创建，由消费者进程消耗。最典型的可消耗资源是用于进程间通信的消息。

执行顺序 1：每个进程都先产生资源，再请求；不会死锁：因为请求时资源都产生了。

b) 进程推进顺序不当



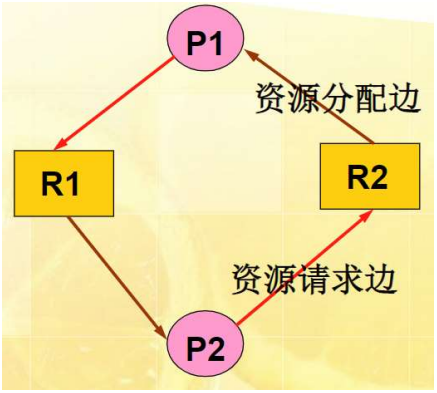


路线④失败原因：当 P1 申请 2 时，P2 需先等 P1 释放 1 后释放 2，但 P1 申请获得 2 后才会释放 1；

产生死锁的必要条件（Coffman 条件）

1. 互斥条件：涉及的资源是非共享的
2. 请求和保持条件：进程在等待新资源时，依然占着已有资源
3. 不剥夺条件：不能强行剥夺进程拥有的资源
4. 环路等待条件：资源分配图存在循环等待链——存在一种进程的循环链，链中的每一个进程已获得的资源同时被链中的下一个进程所请求。

（只要有一个条件不满足，都不会形成死锁）



资源类（资源的不同类型）：用方框表示
资源实例（存在于每个资源中）：用方框中的黑圆点（圈）表示
进程：用圆圈中加进程名表示

分配边：资源实例进程的一条有向边

申请边：进程资源类的一条有向边

处理死锁的基本方法

预防死锁：破坏产生死锁的 4 个必要条件之一

避免死锁：避免系统进入不安全状态

检测与解除死锁：允许系统发生死锁

（3）预防死锁的若干方法及其对系统性能的影响

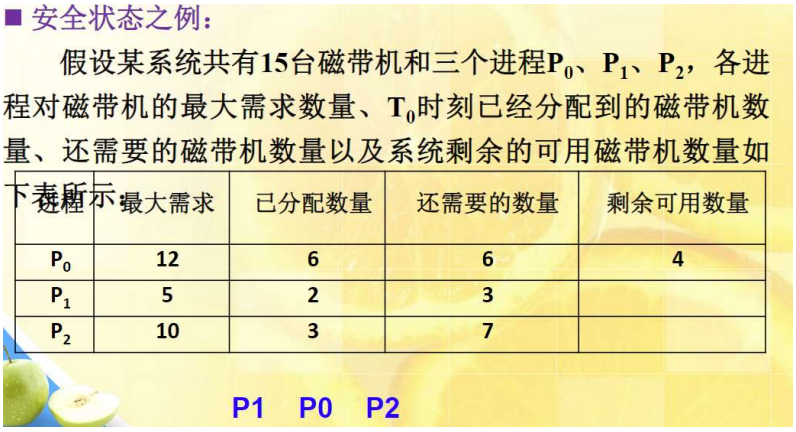
破坏产生死锁的 4 个必要条件之一	具体方法（不包括破坏互斥，互斥是防止发生与时间相关的错误的有力保证，如果被破坏了，会造成结果的不可再现性）	优点	缺点
破坏占有且等待条件	静态分配资源：在作业调度时为选中的作业分配它所需要的**所有资源**，当资源一旦分配给该作业后，在其整个运行期间这些资源**为它独占**。		1）资源利用率低； 2）进程的运行可能被推迟，甚至产生“饥饿” （其中，饥饿（STARVATION）指一个进程一直得不到

			资源。 死锁和饥饿都是由于进程竞争资源而引起的。饥饿一般不占有资源，死锁进程一定占有资源。）
	要求进程在没有资源时才可申请资源：允许**动态申请**资源；获得运行初期所需要的资源后就可以开始执行；运行过程必须释放之前已占有的全部资源后才可以申请；	相比静态分配资源，可以在一定程度提高资源利用率，也可以减少进程启动运行的延迟	
破坏不可剥夺条件	进程申请资源 R 时，有则分配；若没有则需释放其所占有的全部资源后进入阻塞状态。		
	进程申请资源 R 时，有则分配；若没有则进一步判断是否能从其他进程那里进行剥夺；若不能剥夺则进入阻塞状态。		
破坏循环等待条件	采用按序分配资源策略： ✧ 将所有的系统资源按类型进行线性排队，并赋予不同的序号； ✧ 所有进程对资源的请求应严格按资源序号递增顺序提出。	较之前面两种方法提高了资源利用率和系统吞吐量	(1) 降低了添加新设备的灵活性； (2)作业使用资源的顺序与系统规定的申请顺序不同时，会降低资源的利用率

（4）应用银行家算法避免死锁、判断系统的安全性及其是否能满足进程的资源请求

Ⅰ判断系统的安全性

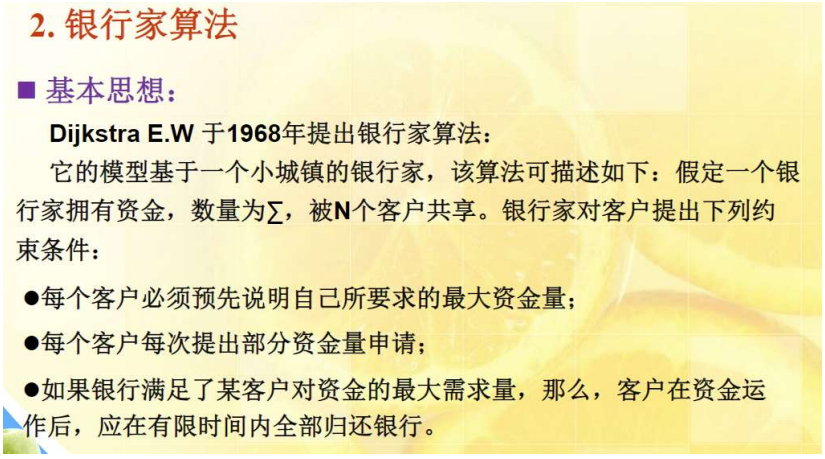
安全状态：设系统中有 n 个进程，若存在一个进程序列<P1，P2，…，Pn>。使得进程 Pi （i=1,2，…，n）以后还需要的资源可以通过系统现有空闲资源加上所有 Pj（j<i）已占有的资源来满足，则称此时系统处于安全状态，进程序列<P1，P2，…，Pn>称为安全序列，因为各进程至少可以按照安全序列中的顺序依次执行完成。如果系统无法找到这样一个安全序列，则称系统处于不安全状态。



当剩余可用数量>还需要的数量，则可以分配给对应进程，并结束后回收其已分配数量；

Ⅱ银行家算法避免死锁

HTTPS://BLOG.CSDN.NET/AI977313677/ARTICLE/DETAILS/72780178



■银行家算法中的数据结构

① 可利用资源向量**Available[m]**。

其中的每一个元素代表一类可利用的资源数目

Available[j]=K，则表示系统中现有R_j类资源K个。

② 最大需求矩阵**Max[1..n,1..m]**

该矩阵定义了系统中n个进程对m类资源的最大需求。

Max[i,j]=K，则表示进程i需要R_j类资源的最大数目为K。

■ 银行家算法中的数据结构

③ 分配矩阵**Allocation[1..n,1..m]**

该矩阵表示系统中每个进程当前已分配到的每类资源数量。

Allocation[i,j]=K，表示进程i当前已分得R_j类资源的数目为K。

④ 需求矩阵**Need[1..n,1..m]**

该矩阵表示每个进程尚需的各类资源数。**Need[i,j]=*K***，则表示进程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]

⑤ 请求向量**Requesti[m];**

某进程提出的资源请求向量。

■ 银行家算法描述

当进程P_i提出资源申请**Requesti[m]**时，系统执行下列步骤：

⑴ 若**Request[i]≤Need[i]**,转⑵；否则错误返回；

⑵ 若**Request[i]≤Available**，转⑶；否则，表示尚无足够资源，P_i须等待；

⑶系统尝试把资源分配给进程P_i，并修改以下数据结构：

Available:= Available-Request[i];

Allocation[i]:= Allocation[i]+Request[i];

Need[i]:= Need[i]-Request[i];

⑷ 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，则将资源分配给进程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。

其中，安全性算法：

■ 安全性算法描述

① 设置两个临时向量：

● 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，它含有*m*个元素，在执行安全算法开始时，Work=Available；

● Finish[n]：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。

■ 安全性算法描述

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

■ 银行家算法举例

例1：假定系统中有4个进程P_0、P_1、P_2、P_3，3类资源R_1、R_2、R_3，数量分别为9、3、6，在T_0时刻的资源分配情况如图所示。

资源 / 进程	Max	Allocation	Need	Available
	R1 R2 R3	R1 R2 R3	R1 R2 R3	R1 R2 R3
P_0	3 2 2	1 0 0	2 2 2	1 1 2
P_1	6 1 3	5 1 1	1 0 2	
P_2	3 1 4	2 1 1	1 0 3	
P_3	4 2 2	0 0 2	4 2 0	

(1)判断T0时刻的安全性

资源 / 进程	Work			Need			Allocation			Work+Allocation			Finish
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3	
P_1	1	1	2	1	0	2	5	1	1	6	2	3	true
P_0	6	2	3	2	2	2	1	0	0	7	2	3	true
P_2	7	2	3	1	0	3	2	1	1	9	3	4	true
P_3	9	3	4	4	2	0	0	0	2	9	3	6	true

由于T_0时刻存在安全序列{P_1，P_0，P_2，P_3},故此时系统是安全的。

■ 银行家算法举例

资源 / 进程	Max			Allocation			Need			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P_0	3	2	2	1	0	0	2	2	2	1	1	2
P_1	6	1	3	5	1	1	1	0	2	0	1	1
P_2	3	1	4	6	1	2	0	0	1			
P_3	4	2	2	0	0	2	4	2	0			

（2）P_1请求资源：$Request_1$（1，0，1）时：

① $Request_1$（1，0，1）≦ $Need_1$（1，0，2）

② $Request_1$（1，0，1）≦ Available（1，1，2）

③试分配资源后，修改数据结构。

④对试分配后状态进行安全性检查：

资源 / 进程	Max			Allocation			Need			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P_0	3	2	2	1	0	0	2	2	2	0	1	1
P_1	6	1	3	6	1	2	0	0	1			
P_2	3	1	4	2	1	1	1	0	3			
P_3	4	2	2	0	0	2	4	2	0			

资源 / 进程	Work			Need			Allocation			Work+Allocation			Finish
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3	
P_1	0	1	1	0	0	1	6	1	2	6	2	3	true
P_0	6	2	3	2	2	2	1	0	0	7	2	3	true
P_2	7	2	3	1	0	3	2	1	1	9	3	4	true
P_3	9	3	4	4	2	0	0	0	2	9	3	6	true

由于此时存在安全序列{P1，P0，P2，P3},故系统是安全的，可为P_1分配上述资源。

■ 银行家算法举例

资源 / 进程	Max			Allocation			Need			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P_0	3	2	2	1	0	0	2	2	2	0	1	1
P_1	6	1	3	6	1	2	0	0	1			
P_2	3	1	4	2	1	1	1	0	3			
P_3	4	2	2	0	0	2	4	2	0			

（3）P_0请求资源：$Request_0$（1，0，1）：

① $Request_0$（1，0，1） ≦ $Need_0$（2，2，2）成立；

② $Request_0$（1，0，1） ≦ Available（0，1，1）不成立，故让P_0等待。

资源 / 进程	Max			Allocation			Need			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P_0	3	2	2	1	0	0	2	2	2	0	1	1
P_1	6	1	3	6	1	2	0	0	1	0,	1,	0
P_2	3	1	4	2	1	1	1	0	3			
P_3	4	2	2	2,	1,	2	1,	0,	2			

（4）P_2请求资源：$Request_2$（0，0，1）时：

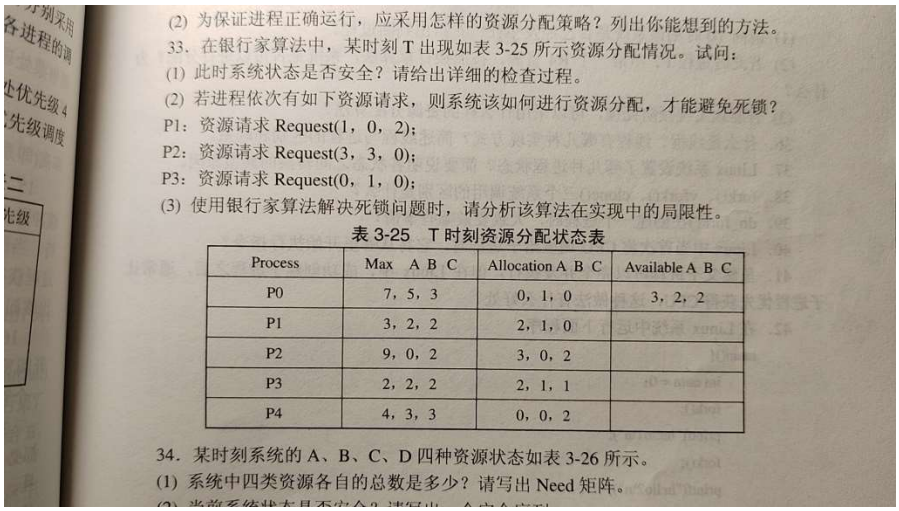
① $Request_2$（0，0，1）≦ $Need_2$（1，0，3）成立；

② $Request_2$（0，0，1）≦ Available（0，1，1）成立；

③试分配资源后，修改数据结构。

④对试分配后的状态进行安全性检查：由于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，所以不能为P_0分配资源，而应恢复原来的状态，让P_0等待。

即当前 AVAILABLE 可以满足对应进程的 NEED，则可以加上对应进程的 ALLOCATION；



(2) 为保证进程正确运行，应采用怎样的资源分配策略？列出你能想到的方法。

33. 在银行家算法中，某时刻 T 出现如表 3-25 所示资源分配情况。试问：

(1) 此时系统状态是否安全？请给出详细的检查过程。

(2) 若进程依次有如下资源请求，则系统该如何进行资源分配，才能避免死锁？

P1：资源请求 Request(1，0，2)；

P2：资源请求 Request(3，3，0)；

P3：资源请求 Request(0，1，0)；

(3) 使用银行家算法解决死锁问题时，请分析该算法在实现中的局限性。

表 3-25　T 时刻资源分配状态表

Process	Max　A B C	Allocation A B C	Available A B C
P0	7，5，3	0，1，0	3，2，2
P1	3，2，2	2，1，0	
P2	9，0，2	3，0，2	
P3	2，2，2	2，1，1	
P4	4，3，3	0，0，2	

34. 某时刻系统的 A、B、C、D 四种资源状态如表 3-26 所示。

(1) 系统中四类资源各自的总数是多少？请写出 Need 矩阵。

《计算机操作系统》（2018 年版）P153 33 题

课堂小组讨论：

教材P153：第33题

解答：（1）系统安全，因为存在安全序列（P1,P3,P0,P2,P4），判断过程略。

（2）系统要想避免死锁，就必须保证每次分配后都能得到安全序列，否则就拒绝分配。根据这一原则，对于进程的请求应考虑分配以后是否安全，若不安全，则不能进行此次分配。题目中有3个请求，按照顺序依次考虑，先考虑能否满足P1，分配后系统仍处于安全状态，因此存在安全序列（P1,P3,P2,P0,P4）。满足P1的请求之后，剩余资源为（2,2,0），对于P4的请求，由于系统没有那么多剩余资源，因此无法满足，系统拒绝P4的请求。最后考虑P0的请求，如果满足P0，分配后剩余资源为（2,1,0），可以找到安全序列（P1,P3,P0,P2,P4），因此可以满足P0的请求。

课堂小组讨论：

（3）银行家算法的局限性：1）系统中进程数量及资源数量众多，而且每当进程申请资源时都要运行该算法，因此算法本身的运行开销大；2）银行家算法是按照最严苛的条件判断安全性的，过于保守，会降低资源利用率及延迟进程推进速度；3）银行家算法在寻找安全序列时完全没有考虑进程之间的同步关系：相互协作进程间的前驱后继关系，这在实际系统中是很不现实的。

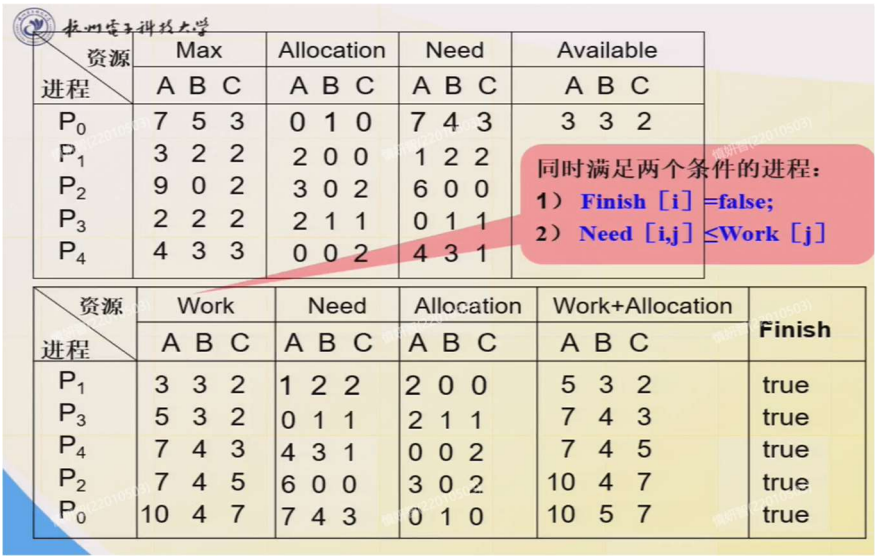
雨课堂题目:

杭州电子科技大学

课堂小组讨论：10分钟

在银行家算法中，某时刻T出现如表所示资源分配情况。试问：
（1）此时系统状态是否安全？若安全，给出安全序列。
（2）若P1提出资源请求Request（1,0,2），系统能否进行分配？若能，给出安全序列。
（3）使用银行家算法解决死锁问题时，请分析该算法在实现中的局限性。

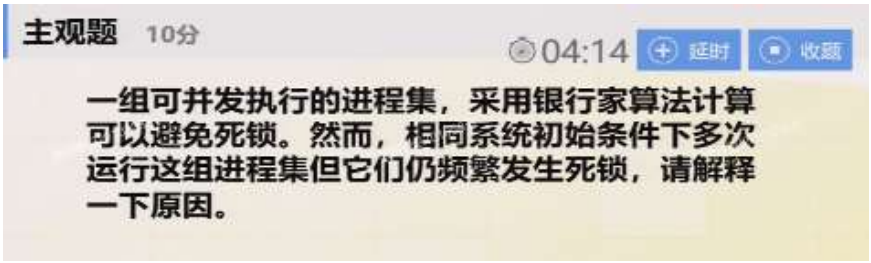
Process	Max			Allocation			Available		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	3	2	2
P1	3	2	2	2	1	0			
P2	9	0	2	3	0	2			
P3	2	2	2	2	1	1			
P4	4	3	3	0	0	2			



资源 / 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	

同时满足两个条件的进程：
1） Finish［i］=false;
2） Need［i,j］≤Work［j］

资源 / 进程	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

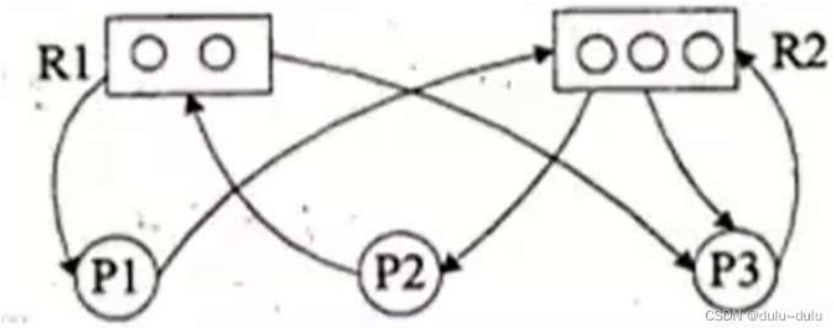


老师答：因为它是理想的方法，并发执行过程中不一定会按照银行家算法对应的顺序执行。

(5) 分析说明复杂的资源分配图的含义

1.资源分配图的概念

资源分配图表示进程和资源之间的请求关系，例如下图：

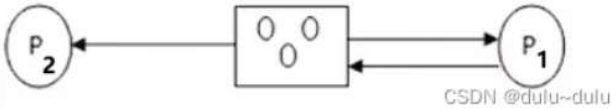


P代表进程，R代表资源，R方框中 有几个圆球就表示有几个这种资源，在图中，**R1指向P1，表示R1已经分配了一个资源给P1了，R1指向P1的边叫做分配边（资源-->进程）；P1指向R2，表示P1还需要一个R2才能执行，P1指向R2的边叫做请求边（进程-->资源）。**

阻塞节点：某进程中所请求的资源已全部分配完毕，无法获取所需资源，则该进程被阻塞了无法继续执行，如上图P2。

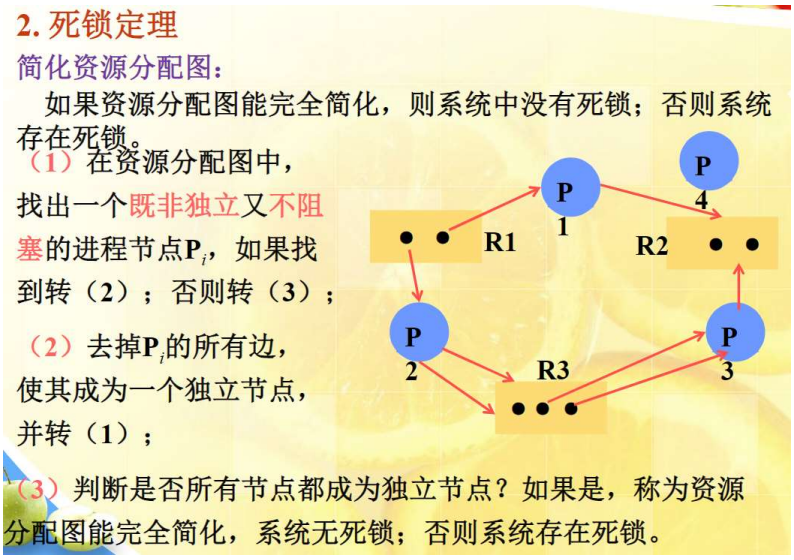
非阻塞节点：某进程所请求的资源还有剩余，可以分配给该进程继续运行。如上图中P1，P3。 **当一个进程资源图中所有进程都是阻塞节点时，即进入死锁状态。**

说明一下：



> 上面的图表示，系统分配一个 R1 资源给进程 p2，然后又分配一个 R1类资源给进程 p1，
> 最后**进程 p1 收到一个 R1 类资源后又继续申请1个R1类资源**，此时，还剩下一个 R1类资源可以分配给 P1，但还没分配给 P1。(注意：图中P1的申请是还没得到响应的，不要以为 R1 指向P1的那个箭头是响应 P1的申请，而分配了资源给 P1)。**“右箭头”跟“左箭头”是没任何关系的。**
> 也就是先分配，再看进程的请求是否能够被满足。如果某个进程的请求能被满足，那么这个进程就是非阻塞节点，不能被满足，就是阻塞节点。

PPT 例题如下：



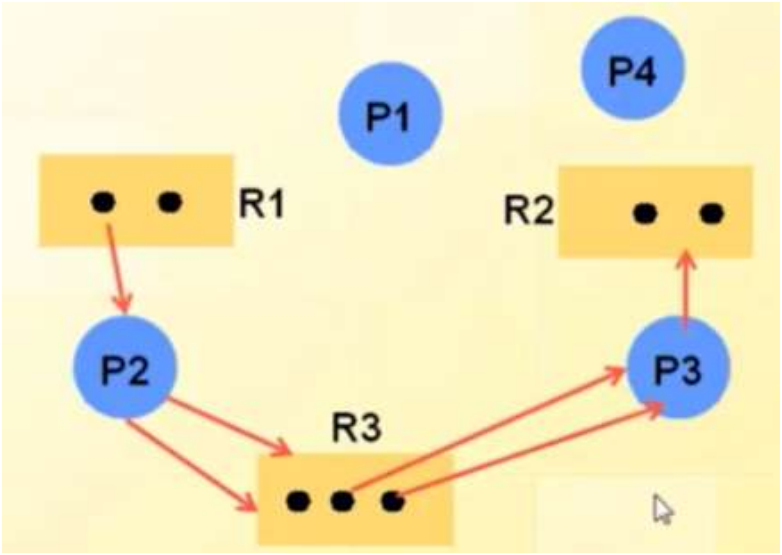
注：在（2）中去掉边以后，原来对应节点所占有的资源会释放；

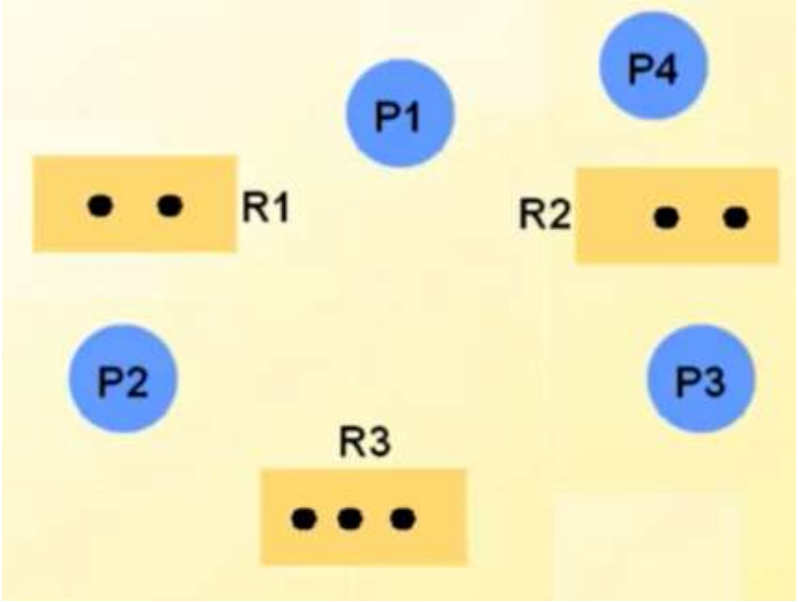
【死锁定理】死锁状态的充分条件是：当且仅当资源分配图是不可完全简化的。

1.先将分配给进程，再看进程的请求（顺序一定不能乱）。

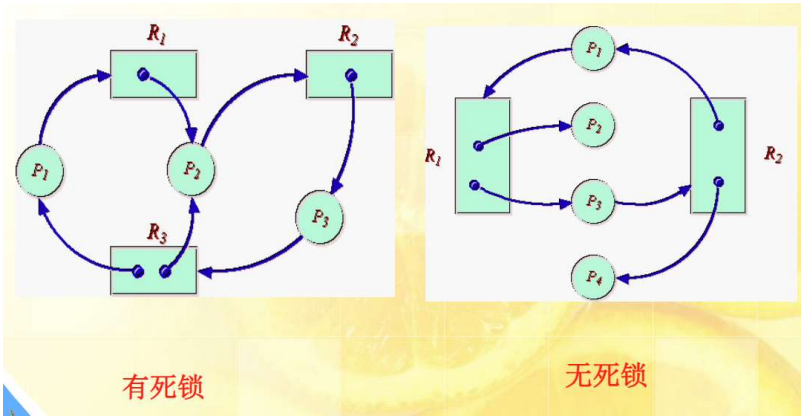
2.对于较复杂的资源分配图，当一个进程是非阻塞节点时，可以想将它“孤立”起来。

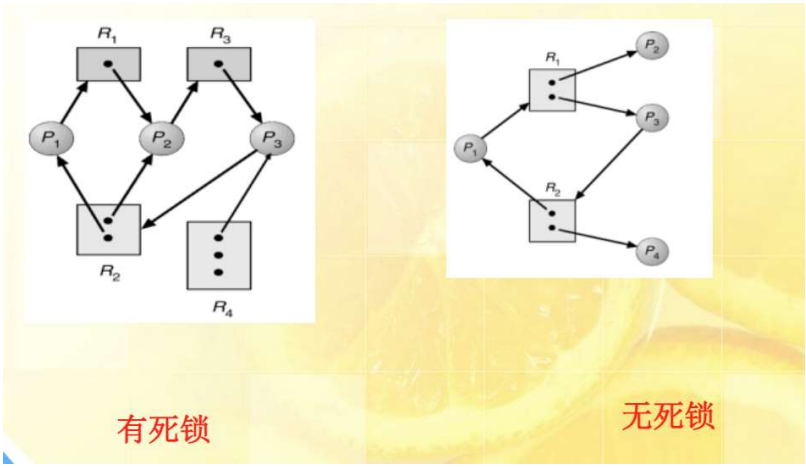
3.进程请求资源才可能发生死锁，所以只有分配边没有请求边的进程节点可以直接“孤立”起来。





（6）根据一个复杂的资源分配图判断当前系统是否出现死锁，能分析说明死锁检查算法思路，并能编程实现（死锁定理及死锁检测算法的思路及编程实现）





参考资料：https://blog.csdn.net/wljliujuan/article/details/79614019（含产生死锁的编程实现）

死锁检测算法：Coffman 算法（与银行家算法类似的数据结构）

（7）根据实际应用场景确定合理的死锁检测时机：根据系统的负载、资源状况等实际情况给出合理的死锁检测时机，以获得较好的整体性能

l 死锁检测时机

（综合考虑以下因素：死锁出现的频繁程度；发生死锁时受到死锁影响的进程数量；进行死锁检测的系统开销）

1. 每当有进程请求资源且得不得满足时就做检测
2. 周期性定时检测
3. 依据 CPU 利用率确定是否进行检测

（8）根据实际应用场景设计合理的死锁解除方法：多种解除死锁的方法，并能分析所给出方法对系统性能的影响

l 死锁的解除

①撤销进程

✧ 撤销所有死锁进程

✧ 一次只撤销一个进程直到解除死锁为止

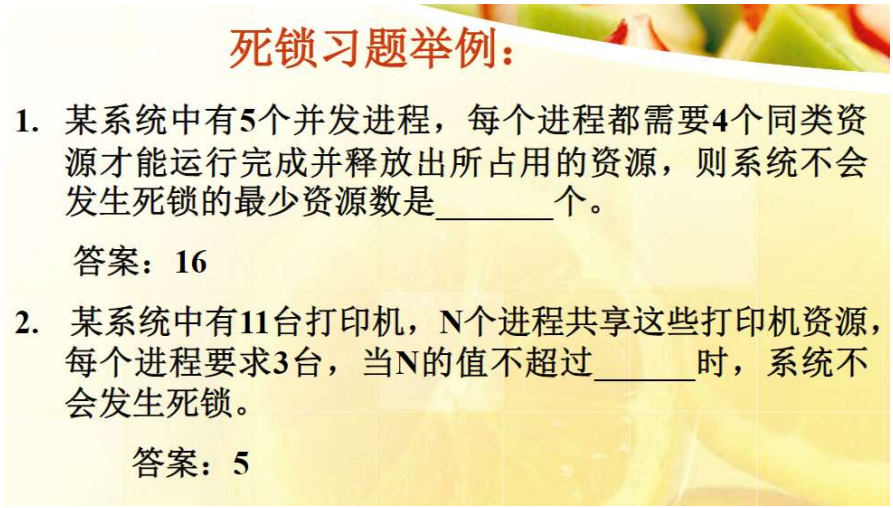
- ➢ 撤销的进程数最少
- ➢ 撤销代价最小
  - •进程优先数；（通常低优先级比高优先级撤销的代价要小）
  - •进程类的外部代价；
  - •进程运行代价

②抢占资源

✧ 选择被抢占资源的进程

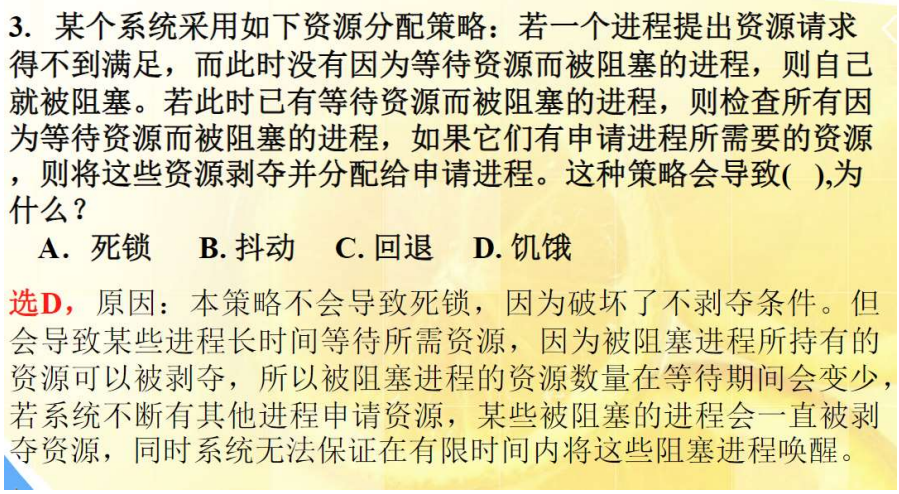
✧ “回滚”问题

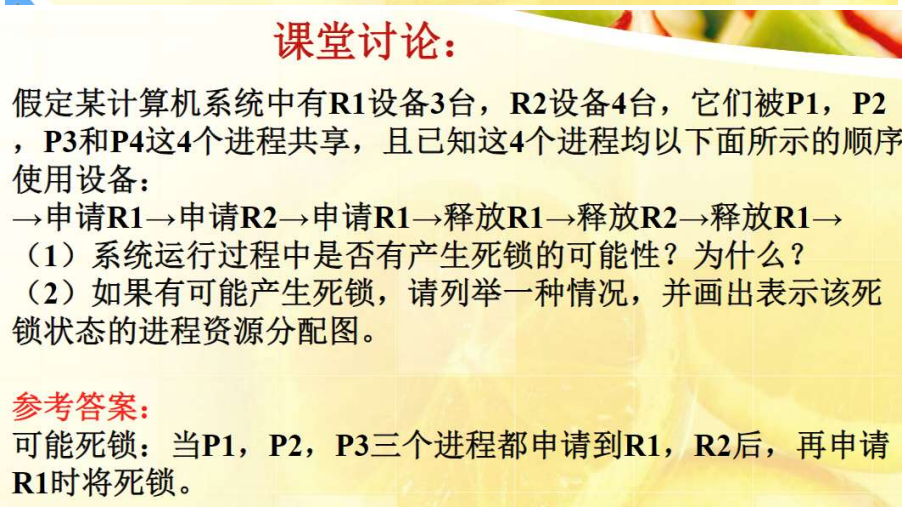
✧ 避免“饥饿”现象



1. 5×（4-1）+1=16

2.（11-1）/（3-1）=5





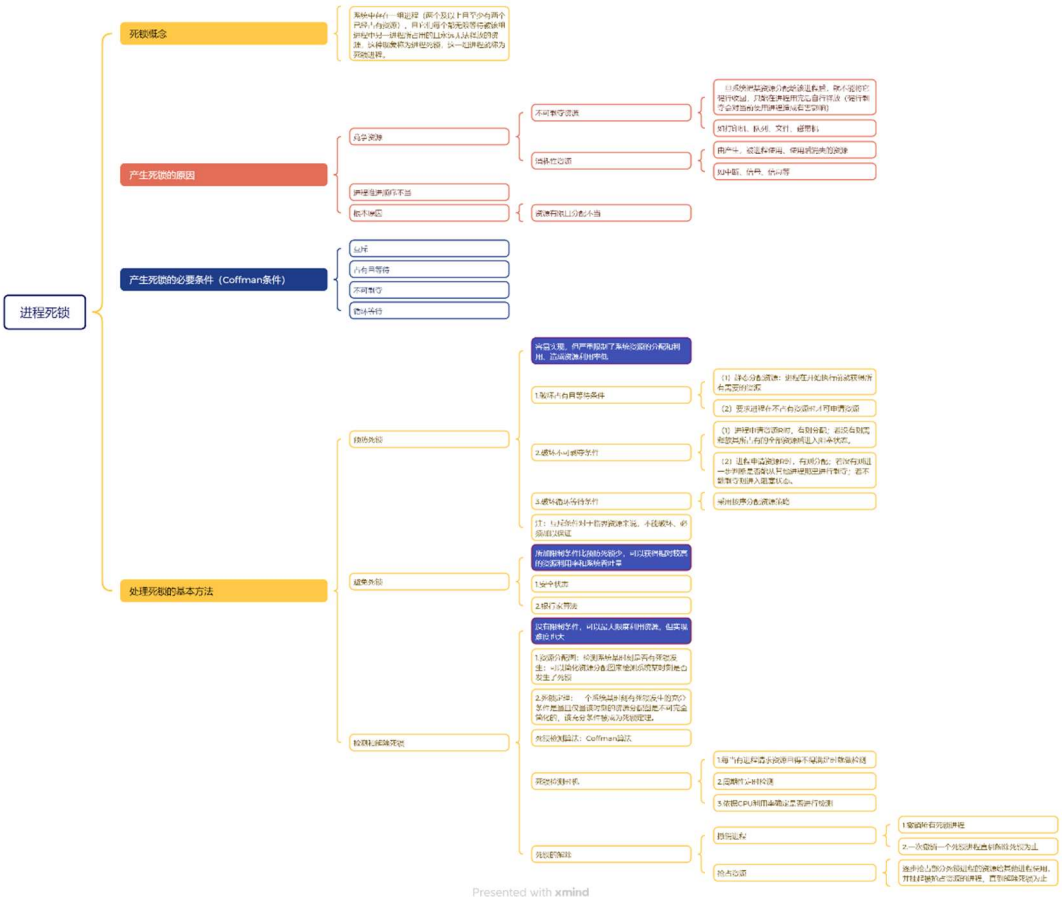
PPT 参考资料

死锁：

https://baike.baidu.com/item/%E6%AD%BB%E9%94%81/2196938?fromtitle=%E8%BF%9B%E7%A8%8B%E6%AD%BB%E9%94%81&fromid=8897805

进程死锁：
https://blog.csdn.net/weixin_40423553/article/details/78879122
死锁及死锁的四个必要条件与处理：策略
https://blog.csdn.net/wljliujuan/article/details/79614019
关于理解资源分配图进行训练的例题：
https://blog.csdn.net/weixin_69884785/article/details/139144181

二、结合课本，对上述内容进行框架梳理：



三、平台测验有关习题——处理死锁的方法-课堂测试

设系统中仅有一类数量为M的独占型资源，系统中有N个进程竞争该类资源，其中每个进程对该类资源的最大需求数量为W，当M、N、W分别去下列值时，可能出现死锁的情况是（）。

- A.M-=2,N=2,W=1
- B.M-=3,N=2,W=2
- C.M-=5,N=3,W=2
- D.M-=6,N=3,W=3

我的答案: D

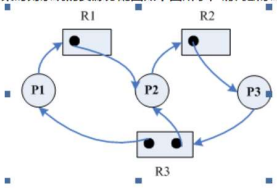
得分: 10

题目解析: 当N*（W-1）+1≤M时，系统不会发生死锁

资源的有序分配策略可以破坏死锁的（）条件。

- A.互斥
- B.请求和保持
- C.不剥夺
- D.循环等待

某时刻系统的资源分配图如下图所示，请判断系统是否发生死锁。



- A.没有发生死锁
- B.P1和P2发生死锁
- C.P2和P3发生死锁
- D.P1、P2和P3都发生死锁

死锁定理是用于处理死锁的（）方法

- A.预防死锁
- B.避免死锁
- C.检测死锁
- D.解除死锁

如果系统中只有一个临界资源，同时有很多进程要竞争该资源，那么系统（）发生死锁。

A.一定会

B.一定不会

在某一时刻，进程P1和P2已执行或将执行下列关于临界资源的操作序列：P1已申请到资源S1，申请资源S2，释放资源S1；P2已申请到资源S2，申请资源S1，释放资源S2，系统继续并发执行P1和P2。若进程申请不到所需要的资源时阻塞，则系统将（）。

A.必定产生死锁

B.可能产生死锁

C.不会产生死锁

D.无法确定是否会产生死锁