2.4死锁

死锁

在第二章中，我们已经涉及到死锁的概念。例如，系统中只有一台扫描仪 R_1 和一台刻录机 R_2。有两个进程 P_1 和 P_2，它们都准备将扫描的文挡刻录到 CD 光盘上，进程 P_1 先请求扫描仪 R_1 并获得成功，进程 P_2 先请求 CD 刻录机 R_2 也获得成功。后来 P_1 又请求 CD 刻录机，因它已被分配给了 P_2 而阻塞。P_2 又请求扫描仪，也因被分配给了 P_1 而阻塞，此时两个进程都被阻塞，双方都希望对方能释放出自己所需要的资源，但它们谁都因不能获得自己所需的资源去继续运行，从而无法释放出自己占有的资源，并且一直处于这样的僵持状态而形成死锁。又如，在第二章的哲学家进餐问题中，如果每一个哲学家因饥饿都拿起了他们左边的筷子，当每一个哲学家又试图去拿起他们右边的筷子时，将会因无筷子可拿而无限期地等待，从而产生死锁问题。在本章的后半部分，我们将对死锁发生的原因、如何预防和避免死锁等问题作较详细的介绍。

死锁的定义

在多道程序系统中，由于多个进程的并发执行，改善了系统资源的利用率并提高了系统的处理能力。然而，多个进程的并发执行也带来了新的问题—死锁。所谓死锁，是指多个进程因竞争资源而造成的一种僵局（互相等待），若无外力作用，这些进程都将无法向前推进。

2014.9 死锁的概念/处理死锁的方法
2015.4 死锁 名词解释

死锁产生的原因

（1）系统资源的竞争

通常系统中拥有的不可剥夺资源，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，如磁带机、打印机等。只有对不可剥夺资源的竞争才可能产生死锁，对可剥夺资源的竞争是不会引起死锁的。

（2）进程推进顺序非法

进程在运行过程中，请求和释放资源的顺序不当，也同样会导致死锁。例如，并发进程 P_1, P_2 分别保持了资源 R_1, R_2，而进程 P_1 申请资源 R_2、进程 P_2 申请资源 R_1 时，两者都会因为所需资源被占用而阻塞，于是导致死锁。

信号量使用不当也会造成死锁。进程间彼此相互等待对方发来的消息，也会使得这些进程间无法继续向前推进。例如，进程 A 等待进程 B 发的消息，进程 B 又在等待进程 A 发的消息，可以看出进程 A 和 B 不是因为竞争同一资源，而是在等待对方的资源导致死锁。

- 进程竞争不可剥夺资源
- 进程推进顺序非法
- 信号量使用 不当

死锁的必要条件

2. 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，但产生进程死锁是必须具备一定条件的。综上所述不难看出，产生死锁必须同时具备下面四个必要条件，只要其中任一个条件不成立，死锁就不会发生：

(1) 互斥条件。进程对所分配到的资源进行排它性使用，即在一段时间内，某资源只能被一个进程占用。如果此时还有其它进程请求该资源，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，而该资源已被其它进程占有，此时请求进程被阻塞，但对自己已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$\{P_0, P_1, P_2, \cdots, P_n\}$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。

直观上看，循环等待条件似乎和死锁的定义一样，其实不然。按死锁定义构成等待环所要求的条件更严，它要求 P_i 等待的资源必须由 P_{i+1} 来满足，而循环等待条件则无此限制。例如，系统中有两台输出设备，P_0 占有一台，P_K 占有另一台，且 K 不属于集合{0, 1, …, n}。P_n 等待一台输出设备，它可从 P_0 获得，也可能从 P_K 获得。因此，虽然 P_n, P_0 和其他一些进程形成了循环等待圈，但 P_K 不在圈内，若 P_K 释放了输出设备，则可打破循环等待，如图 2.14 所示。因此循环等待只是死锁的必要条件。

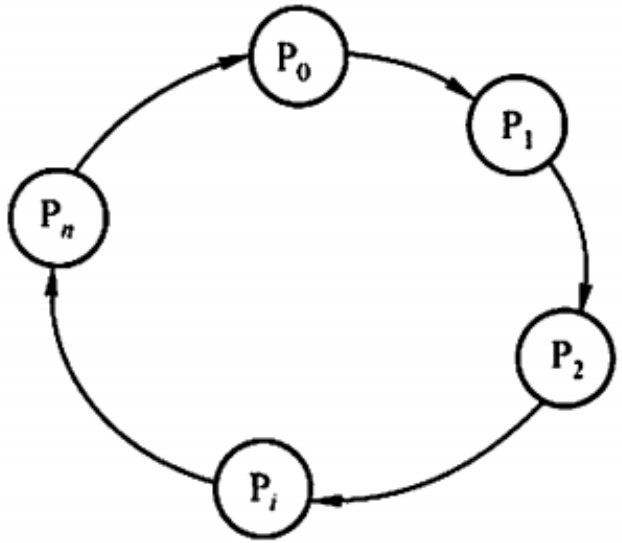

图 2.13　循环等待

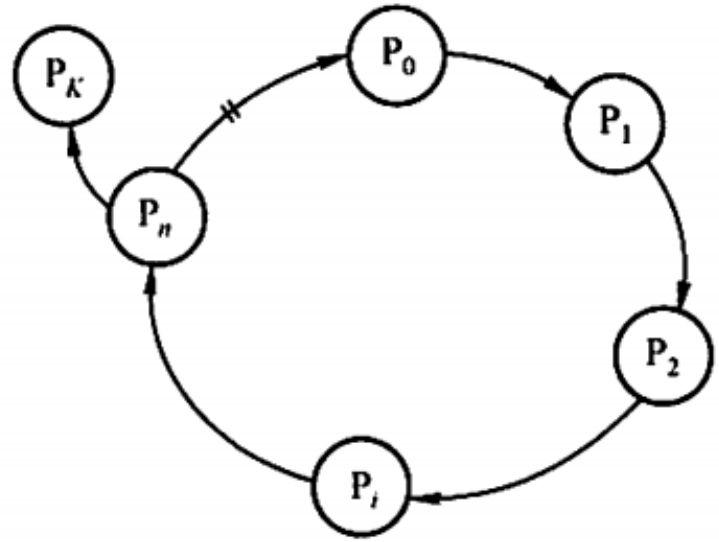

图 2.14　满足条件但无死锁

死锁的处理方法

3. 处理死锁的方法

目前处理死锁的方法可归结为四种：

(1) 预防死锁。这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，去破坏产生死锁四个必要条件中的一个或几个来预防产生死锁。预防死锁是一种较易实现的方法，已被广泛使用。

(2) 避免死锁。同样是属于事先预防策略，但它并不是事先采取各种限制措施，去破坏产生死锁的四个必要条件，而是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，从而可以避免发生死锁。

(3) 检测死锁。这种方法无须事先采取任何限制性措施，允许进程在运行过程中发生死锁。但可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，把进程从死锁中解脱出来。

(4) 解除死锁。当检测到系统中已发生死锁时，就采取相应措施，将进程从死锁状态中解脱出来。常用的方法是撤消一些进程，回收它们的资源，将它们分配给已处于阻塞状态的进程，使其能继续运行。

上述的四种方法，从(1)到(4)对死锁的防范程度逐渐减弱，但对应的是资源利用率的提高，以及进程因资源因素而阻塞的频度下降(即并发程度提高)。

死锁的处理方法

- 预防死锁 破坏产生死锁的4个必要条件中的一个或多个
- 避免死锁 使用某种方法防止系统进入不安全状态
- 死锁的检测和解除 不采取任何措施，在发生死锁后，系统能够及时检测出死锁，然后采取某种措施接触死锁

1.死锁预防

2．破坏不剥夺条件

当一个已保持了某些不可剥夺资源的进程请求新的资源而得不到满足时，它必须释放已经保持的所有资源，待以后需要时再重新申请。这意味着，一个进程已占有的资源会被暂时释放，或者说是被剥夺，或从而破坏了不剥夺条件。

该策略实现起来比较复杂，释放已获得的资源可能造成前一阶段工作的失效，反复地申请和释放资源会增加系统开销，降低系统吞吐量。这种方法常用于状态易于保存和恢复的资源，如 CPU 的寄存器及内存资源，一般不能用于打印机之类的资源。

缺点：增加了进程的周转时间；增加了系统开销；降低了吞吐量

预先静态分配法

3．破坏请求并保持条件

采用预先静态分配方法，即进程在运行前一次申请完它所需要的全部资源，在它的资源未满足前，不把它投入运行。一旦投入运行，这些资源就一直归它所有，不再提出其他资源请求，这样就可以保证系统不会发生死锁。

这种方式实现简单，但缺点也显而易见，系统资源被严重浪费，其中有些资源可能仅在运行初期或运行快结束时才使用，甚至根本不使用。而且还会导致“饥饿”现象，由于个别资源长期被其他进程占用时，将致使等待该资源的进程迟迟不能开始运行。

缺点：资源严重浪费，降低了资源利用率；进程会经常发生饥饿现象

顺序资源分配法

4．破坏循环等待条件

为了破坏循环等待条件，可采用顺序资源分配法。首先给系统中的资源编号，规定每个进程必须按编号递增的顺序请求资源，同类资源一次申请完。也就是说，只要进程提出申请分配资源 R_i，则该进程在以后的资源申请中就只能申请编号大于 R_i 的资源。

这种方法存在的问题是，编号必须相对稳定，这就限制了新类型设备的增加；尽管在为资源编号时已考虑到大多数作业实际使用这些资源的顺序，但也经常会发生作业使用资源的顺序与系统规定顺序不同的情况，造成资源的浪费；此外，这种按规定次序申请资源的方法，也必然会给用户的编程带来麻烦。

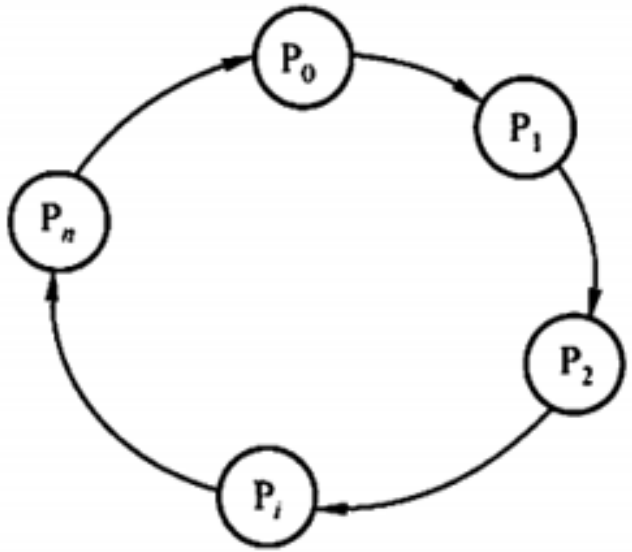

图 2.13　循环等待

缺点：无法增加新类型的设备；实际作业使用资源顺序和系统规定不同，浪费资源；
用户编程麻烦

避免死锁

3.7.1 系统安全状态

在死锁避免方法中，把系统的状态分为安全状态和不安全状态。当系统处于安全状态时，可避免发生死锁。反之，当系统处于不安全状态时，则可能进入到死锁状态。

避免死锁的方法中，允许进程动态地申请资源，但系统在进行资源分配之前，应先计算此次分配的安全性。若此次分配不会导致系统进入不安全状态，则允许分配；否则让进程等待。

所谓安全状态，是指系统能按某种进程推进顺序（$P_1, P_2, \cdots, P_n$）为每个进程 P_i 分配其所需的资源，直至满足每个进程对资源的最大需求，使每个进程都可顺序完成。此时称 $P_1, P_2, \cdots, P_n$ 为安全序列。若系统无法找到一个安全序列，则称系统处于不安全状态。

假设系统中有三个进程 P_1，P_2 和 P_3，共有 12 台磁带机。进程 P_1 共需要 10 台磁带机，P_2 和 P_3 分别需要 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，尚有 3 台未分配，见表 2.5。

表 2.5　资源分配

进程	最大需求	已分配	可用
P_1	10	5	3
P_2	4	2	
P_3	9	2	

在 T_0 时刻是安全的，因为存在一个安全序列 P_2, P_1, P_3，只要系统按此进程序列分配资源，那么每个进程都能顺利完成。也就是说，当前可用磁带机为 3 台，先把 3 台磁带机分配给 P_2 以满足其最大需求，P_2 结束并归还资源后，系统有 5 台磁带机可用；接下来给 P_1 分配 5 台磁带机以满足其最大需求，P_1 结束并归还资源后，剩余 10 台磁带机可用；最后分配 7 台磁带机给 P_3，这样 P_3 也能顺利完成。

若在 T_0 时刻后，系统分配 1 台磁带机给 P_3，系统剩余可用资源数为 2，此时系统进入不安全状态，因为此时已无法再找到一个安全序列。当系统进入不安全状态后，便可能导致死锁。例如，把剩下的 2 台磁带机分配给 P_2，这样，P_2 完成后只能释放 4 台磁带机，既不能满足 P_1 又不能满足 P_3，致使它们都无法推进到完成，彼此都在等待对方释放资源，陷入僵局，即导致死锁。

银行家算法

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。

为实现银行家算法，每一个新进程在进入系统时，它必须申明在运行过程中，可能需要每种资源类型的最大单元数目，其数目不应超过系统所拥有的资源总量。当进程请求一组资源时，系统必须首先确定是否有足够的资源分配给该进程。若有，再进一步计算在将这些资源分配给进程后，是否会使系统处于不安全状态。如果不会，才将资源分配给它，否则让进程等待。

(1) 可利用资源向量 Available。这是一个含有 m 个元素的数组，其中的每一个元素代表一类可利用的资源数目，其初始值是系统中所配置的该类全部可用资源的数目，其数值随该类资源的分配和回收而动态地改变。如果 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。这是一个 n × m 的矩阵，它定义了系统中 n 个进程中的每一个进程对 m 类资源的最大需求。如果 Max[i, j] = K，则表示进程 i 需要 R_j 类资源的最大数目为 K。

(3) 分配矩阵 Allocation。这也是一个 n × m 的矩阵，它定义了系统中每一类资源当前已分配给每一进程的资源数。如果 Allocation[i, j] = K，则表示进程 i 当前已分得 R_j 类资源的数目为 K。

(4) 需求矩阵 Need。这也是一个 n × m 的矩阵，用以表示每一个进程尚需的各类资源数。如果 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。

上述三个矩阵间存在下述关系：

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

一般情况下，在银行家算法的题目中，Max 矩阵和 Allocation 矩阵是已知条件，而求出 Need 矩阵是解题的第一步。

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 $Request_i[j] \leqslant Need[i, j]$，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 $Request_i[j] \leqslant Available[j]$，便转向步骤(3)；否则，表示尚无足够资源，$P_i$ 须等待。

(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）安全性算法

设置工作向量 Work，有 m 个元素，表示系统中的剩余可用资源数目。在执行安全性算法开始时，Work = Available。

① 初始时安全序列为空。

② 从 Need 矩阵中找出符合下面条件的行：该行对应的进程不在安全序列中，而且该行小于等于 Work 向量，找到后，把对应的进程加入安全序列；若找不到，则执行步骤④。

③ 进程 P_i 进入安全序列后，可顺利执行，直至完成，并释放分配给它的资源，因此应执行 Work = Work + Allocation[i]，其中 Allocation[i]表示进程 P_i 代表的在 Allocation 矩阵中对应的行，返回步骤②。

④ 若此时安全序列中已有所有进程，则系统处于安全状态，否则系统处于不安全状态。

看完上面对银行家算法的过程描述后，可能会有眼花缭乱的感觉，现在通过举例来加深理解。

表 2.6　T_0 时刻的资源分配表

进程 \ 资源情况	Max A B C	Allocation A B C	Available A B C
P_0	7 5 3	0 1 0	3 3 2 (2 3 0)
P_1	3 2 2	2 0 0 (3 0 2)	
P_2	9 0 2	3 0 2	
P_3	2 2 2	2 1 1	
P_4	4 3 3	0 0 2	

① 从题目中我们可以提取 Max 矩阵和 Allocation 矩阵，这两个矩阵相减可得到 Need 矩阵：

$$\begin{bmatrix} 7 & 5 & 3 \\ 3 & 2 & 2 \\ 9 & 0 & 2 \\ 2 & 2 & 2 \\ 4 & 3 & 3 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 7 & 4 & 3 \\ 1 & 2 & 2 \\ 6 & 0 & 0 \\ 0 & 1 & 1 \\ 4 & 3 & 1 \end{bmatrix}$$

Max　　Allocation　　Need

$$\begin{bmatrix}7 & 5 & 3\\ 3 & 2 & 2\\ 9 & 0 & 2\\ 2 & 2 & 2\\ 4 & 3 & 3\end{bmatrix}-\begin{bmatrix}0 & 1 & 0\\ 2 & 0 & 0\\ 3 & 0 & 2\\ 2 & 1 & 1\\ 0 & 0 & 2\end{bmatrix}=\begin{bmatrix}7 & 4 & 3\\ 1 & 2 & 2\\ 6 & 0 & 0\\ 0 & 1 & 1\\ 4 & 3 & 1\end{bmatrix}$$

Max　　Allocation　　Need

② 然后，将 Work 向量与 Need 矩阵的各行进行比较，找出比 Work 矩阵小的行。例如，在初始时，

$$(3,3,2)>(1,2,2)$$
$$(3,3,2)>(0,1,1)$$

对应的两个进程分别为 P_1 和 P_3，这里我们选择 P_1（也可以选择 P_3）暂时加入安全序列。

③ 释放 P_1 所占的资源，即把 P_1 进程对应的 Allocation 矩阵中的一行与 Work 向量相加：

$$(3\quad 3\quad 2)+(2\quad 0\quad 0)=(5\quad 3\quad 2)=\text{Work}$$

此时需求矩阵更新为（去掉了 P_1 对应的一行）：

$$\begin{matrix}P_0\\ P_2\\ P_3\\ P_4\end{matrix}\begin{bmatrix}7 & 4 & 3\\ 6 & 0 & 0\\ 0 & 1 & 1\\ 4 & 3 & 1\end{bmatrix}$$

再用更新的 Work 向量和 Need 矩阵重复步骤②。利用安全性算法分析 T_0 时刻的资源分配情况如表 2.7 所示，最后得到一个安全序列{P_1, P_3, P_4, P_2, P_0}。

表 2.7　T_0 时刻的安全序列的分析

资源情况 / 进程	Work			Need			Allocation			Work+Allocation		
	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
P_3	5	3	2	0	1	1	2	1	1	7	4	3
P_4	7	4	3	4	3	1	0	0	2	7	4	5
P_2	7	4	5	6	0	0	3	0	2	10	4	7
P_0	10	4	7	7	4	3	0	1	0	10	5	7

表 2.6　T_0 时刻的资源分配表

进程 \ 资源情况	Max	Allocation	Available
	A B C	A B C	A B C
P_0	7 5 3	0 1 0	3 3 2 (2 3 0)
P_1	3 2 2	2 0 0 (3 0 2)	
P_2	9 0 2	3 0 2	
P_3	2 2 2	2 1 1	
P_4	4 3 3	0 0 2	

$$\begin{bmatrix} 7 & 5 & 3 \\ 3 & 2 & 2 \\ 9 & 0 & 2 \\ 2 & 2 & 2 \\ 4 & 3 & 3 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 7 & 4 & 3 \\ 1 & 2 & 2 \\ 6 & 0 & 0 \\ 0 & 1 & 1 \\ 4 & 3 & 1 \end{bmatrix}$$

Max　　Allocation　　Need

（1）P_1 请求资源：P_1 发出请求向量 $Request_1(1, 0, 2)$，系统按银行家算法进行检查：

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

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

系统先假定可为 P_1 分配资源，并修改

$Available = Available - Request_1 = (2, 3, 0)$

$Allocation_1 = Allocation_1 + Request_1 = (3, 0, 2)$

$Need_1 = Need_1 - Request_1 = (0, 2, 0)$

由此形成的资源变化情况如表 2.6 中的圆括号所示。

令 $Work = Available = (2, 3, 0)$，再利用安全性算法检查此时系统是否安全，如表 2.8 所示。

表 2.8　P_1 申请资源时的安全性检查

资源情况 / 进程	Work			Need			Allocation			Work+Allocation		
	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
P_3	5	3	2	0	1	1	2	1	1	7	4	3
P_4	7	4	3	4	3	1	0	0	2	7	4	5
P_0	7	4	5	7	4	3	0	1	0	7	5	5
P_2	7	5	5	6	0	0	3	0	2	10	5	7

由所进行的安全性检查得知，可找到一个安全序列{P_1, P_3, P_4, P_2, P_0}。因此，系统是安全的，可以立即将 P_1 所申请的资源分配给它。分配后系统中的资源情况如表 2.9 所示。

（2）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：

（3）P_0 请求资源：P_0 发出请求向量 $Requst_0(0, 2, 0)$，系统按银行家算法进行检查：

死锁的检测和解除

1．资源分配图

系统死锁可利用资源分配图来描述。如图 2.15 所示，用圆圈代表一个进程，用框代表一类资源。由于一种类型的资源可能有多个，因此用框中的一个圆代表一类资源中的一个资源。从进程到资源的有向边称为请求边，表示该进程申请一个单位的该类资源；从资源到进程的边称为分配边，表示该类资源已有一个资源分配给了该进程。

在图 2.15 所示的资源分配图中，进程 P_1 已经分得了两个 R_1 资源，并又请求一个 R_2 资源；进程 P_2 分得了一个 R_1 资源和一个 R_2 资源，并又请求一个 R_1 资源。

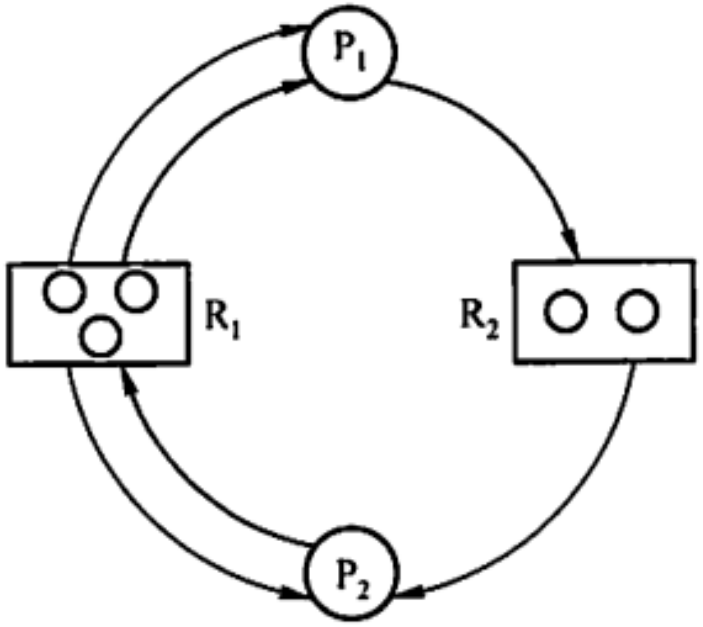

图 2.15　资源分配示例

死锁定理

(1) 在资源分配图中，找出一个既不阻塞又非独立的进程结点 P_i。在顺利的情况下，P_i 可获得所需资源而继续运行，直至运行完毕，再释放其所占有的全部资源，这相当于消去 P_i 的请求边和分配边，使之成为孤立的结点。在图 3-20(a)中，将 P_1 的两个分配边和一个请求边消去，便形成图(b)所示的情况。

(2) P_1 释放资源后，便可使 P_2 获得资源而继续运行，直至 P_2 完成后又释放出它所占有的全部资源，形成图(c)所示的情况，即将 P_2 的两条请求边和一条分配边消去。

(3) 在进行一系列的简化后，若能消去图中所有的边，使所有的进程结点都成为孤立结点，则称该图是可完全简化的；若不能通过任何过程使该图完全简化，则称该图是不可完全简化的。

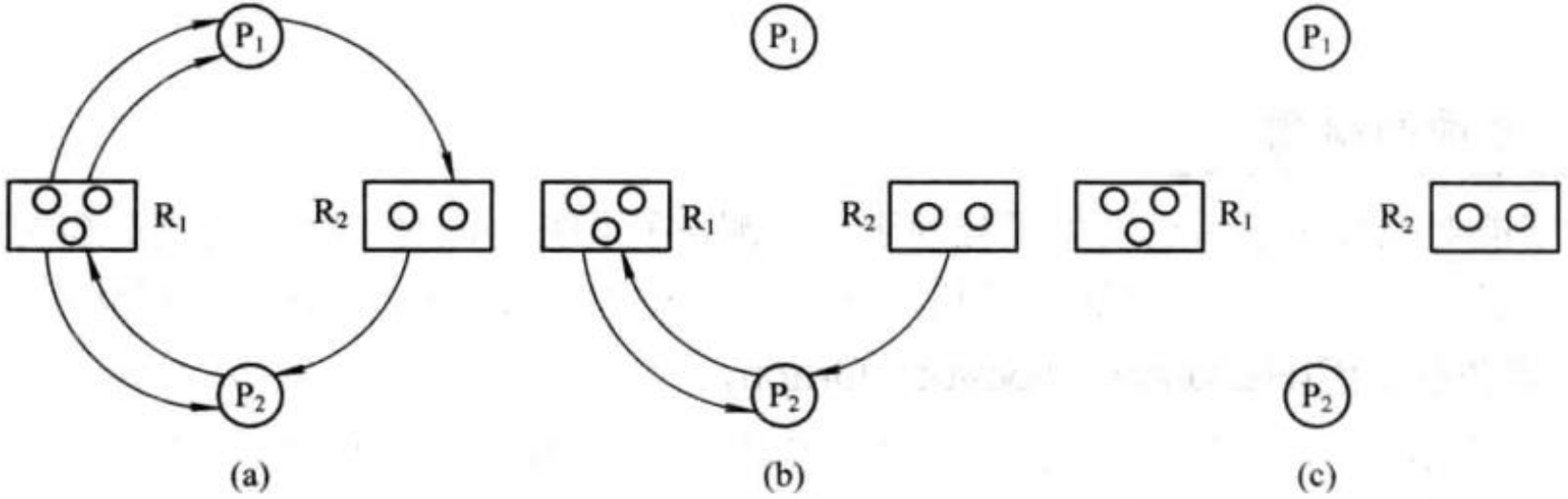

图 3-20　资源分配图的简化

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？有关文献已经证明，所有的简化顺序都将得到相同的不可简化图。同样可以证明：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。该充分条件被称为死锁定理。

死锁的解除

一旦检测出死锁，就应立即采取相应的措施来解除死锁。死锁解除的主要方法有：

1）资源剥夺法。挂起某些死锁进程，并抢占它的资源，将这些资源分配给其他的死锁进程。但应防止被挂起的进程长时间得不到资源而处于资源匮乏的状态。

2）撤销进程法。强制撤销部分甚至全部死锁进程并剥夺这些进程的资源。撤销的原则可以按进程优先级和撤销进程代价的高低进行。

3）进程回退法。让一（或多）个进程回退到足以回避死锁的地步，进程回退时自愿释放资源而非被剥夺。要求系统保持进程的历史信息，设置还原点。

真题

- 2012.3 死锁避免的基本思想及银行家算法
- 2013.2 死锁的判断
- 2014.9 死锁的概念/处理死锁的方法
- 2015.4 死锁 名词解释
- 2016.4 死锁的原因
- 2017.4 处理死锁的基本方法
- 2018.2 死锁 名词解释
- 2018.12
- 2019.13/15
- 2020.4

练习

- 课本预留的例题
- 何谓死锁？产生死锁的原因和必要条件是什么？
- 请详细说明可通过哪些途径预防死锁？
- 在解决死锁问题的几个方法中，那种方法最易于实现？哪种方法使得资源利用率最高？

- 答：解决死锁的方法有预防死锁，避免死锁，检测和解除死锁，其中预防死锁方法最容易实现，但由于所施加的限制条件过于严格,会导致系统资源利用率和系统吞吐量降低；而检测和解除死锁方法使得系统有较好的资源利用率和吞吐量

王道P152

06. 某系统有 R_1, R_2 和 R_3 共三种资源，在 T_0 时刻 P_1, P_2, P_3 和 P_4 这四个进程对资源的占用和需求情况见下表，此时系统的可用资源向量为(2, 1, 2)。试问：

1）系统是否处于安全状态？若安全，则请给出一个安全序列。

2）若此时进程 P_1 和进程 P_2 均发出资源请求向量 Request(1, 0, 1)，为了保证系统的安全性，应如何分配资源给这两个进程？说明所采用策略的原因。

资源情况 / 进程	最大资源需求量			已分配资源数量		
	R_1	R_2	R_3	R_1	R_2	R_3
P_1	3	2	2	1	0	0
P_2	6	1	3	4	1	1
P_3	3	1	4	2	1	1
P_4	4	2	2	0	0	2

3）若 2）中两个请求立即得到满足后，系统此刻是否处于死锁状态？