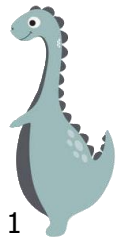
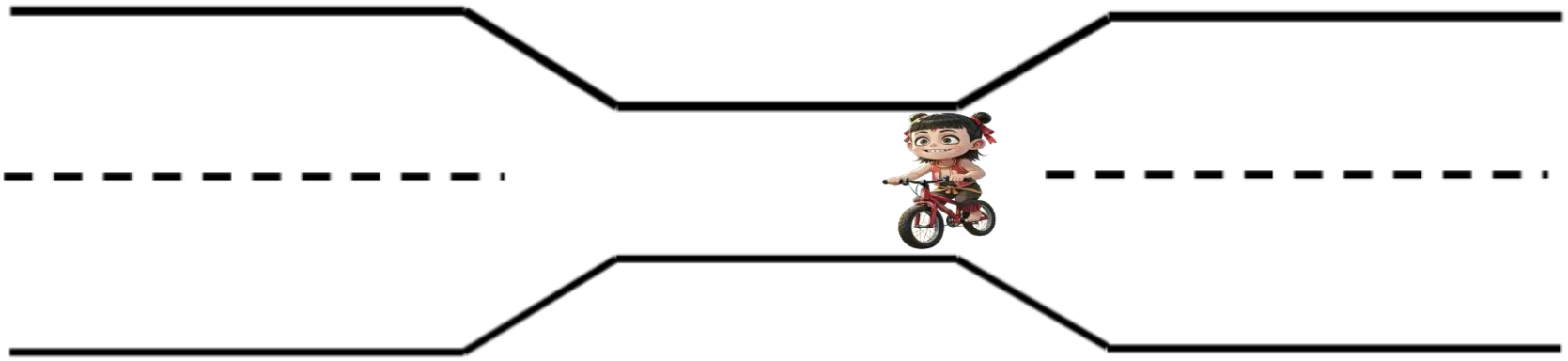
第6章 死锁

- 本章内容
- 1. 死锁的定义、产生的原因、必要条件
- 2. 解决死锁的典型方法：死锁预防、死锁避免、死锁检测、死锁解除
- 3. 以哲学家进餐问题为例，了解导致死锁的原因，以及怎样解决死锁



死锁



- 桥梁只能单向通行
- 桥的每个部分可视为一个资源
- 可能发生饥饿
  - 由于一个方向的持续车流，另一个方向的车辆无法通过桥梁

死锁示例

- 线程A和线程B共享两把全局互斥锁A和B

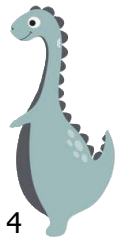
```
void proc_A(void){
   lock(A);
   // t0时刻
   lock(B);
   // 临界区
   unlock(B);
   unlock(A);
}
```

```
void proc_B(void){
   lock(B);
   // t0时刻
   lock(A);
   // 临界区
   unlock(A);
   unlock(B);
}
```

- 假设在t_0时刻，线程A获得了锁A，将要尝试获得锁B。线程B获得了锁B，正等待获取锁A
- 两个线程都无法获得对方持有的锁，无法进入临界区，此时的状态我们称之为**死锁**

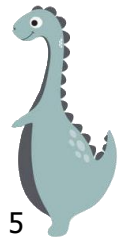
死锁定义

- 当有多个（两个及以上）线程为有限资源竞争时，有的线程就会因为在某一时刻没有可用的空闲资源而陷入等待。死锁(deadlock)就是指这一组中的每个线程都在等待组内其他线程释放资源从而造成的无限等待。若无外力作用，这些进程将永远都不能向前推进。
- 注意：
  - 死锁是因资源竞争造成的僵局
  - 通常死锁至少涉及两个进程
  - 死锁与部分进程及资源相关



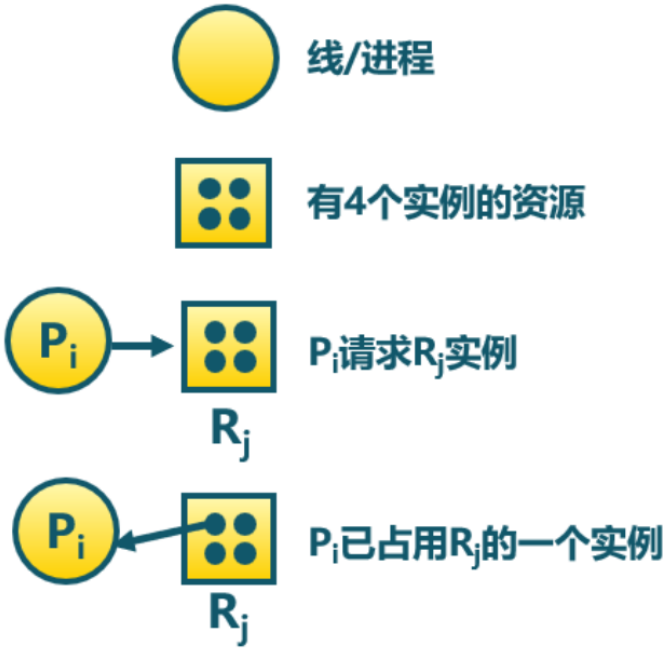
系统模型

- 系统有m类资源$R_1, R_2, \ldots, R_m$
  - CPU周期，内存空间，I/O设备
- 每一类资源R_i有W_i个实例
- 进程按如下方式利用资源
  - 申请 request
  - 使用 use
  - 释放 release



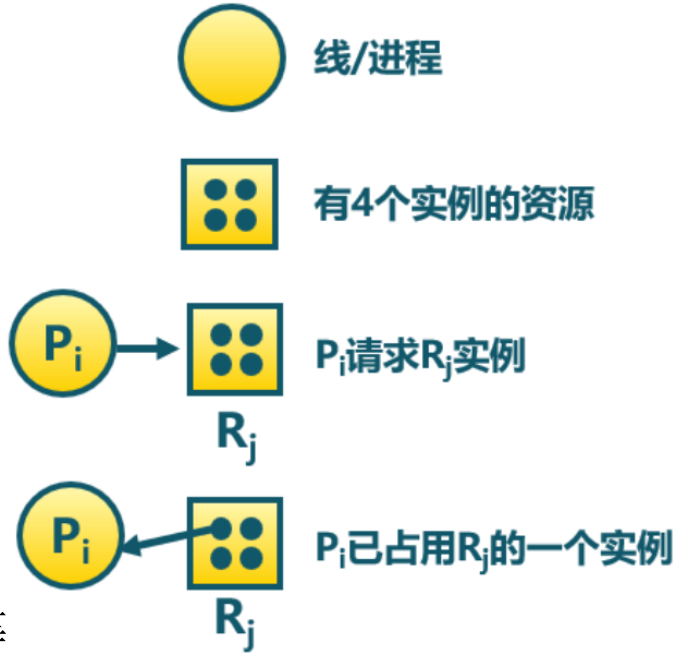
资源分配图

- 资源分配图由一个节点集合V和一个边集合E组成
  - V被分为两个部分
    - 系统中全部进程集合P={P1, P2, …, Pn}
    - 系统中全部资源集合R= {R1, R2, …, Rm}
  - 请求边：有向边$P_i \rightarrow R_j$;
  - 分配边：有向边 $R_j \rightarrow P_i$
- 在图中，用圆形表示进程P_i，用方框表示资源类型R_j，在方框中用圆点表示实例



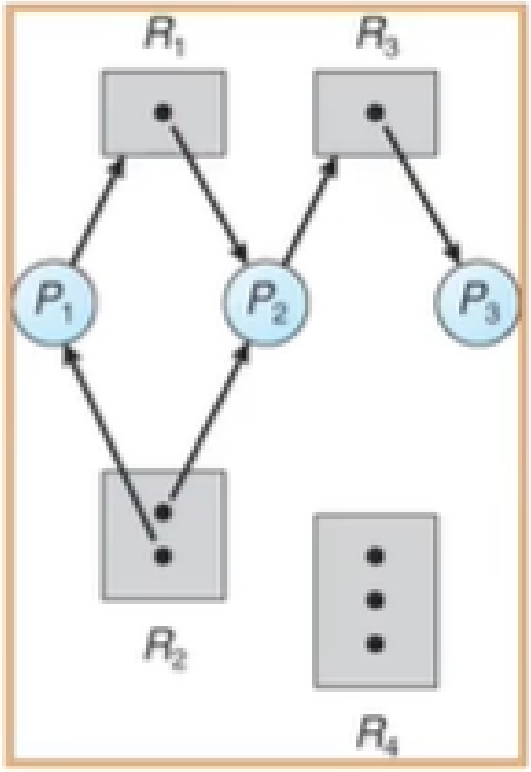
资源分类

- 可重用资源（Reusable Resource）
  - 任何时刻只能有一个线/进程使用资源
  - 资源被释放后，其他线/进程可重用
  - 可重用资源示例
    - 硬件：处理器、内存、设备等
    - 软件：文件、数据库和信号量等
  - 可能出现死锁：每个进程占用一部分资源并请求其它资源
- 可消耗资源(Consumable resource)
  - 资源可被销毁
  - 可消耗资源示例
    - 在I/O缓冲区的中断、信号、消息等
  - 可能出现死锁：进程间相互等待接收对方的消息



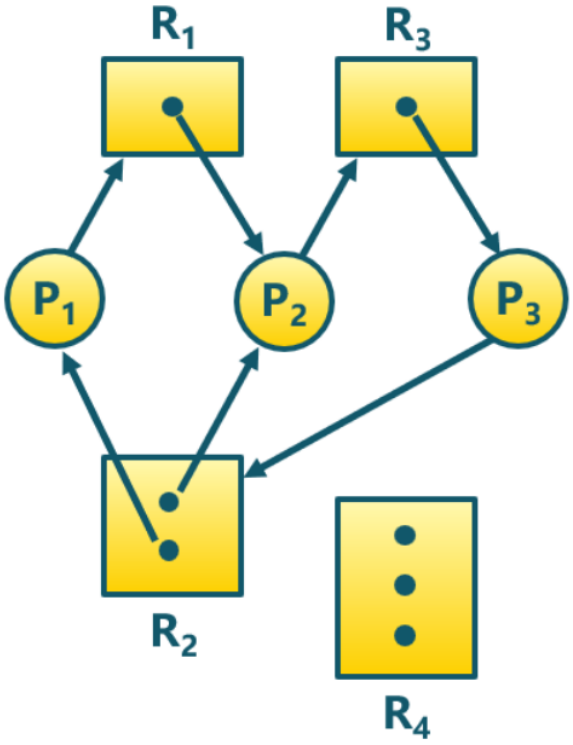
资源分配图示例

- R_1有一个实例
- R_2 有两个实例
- R_3 有一个实例
- R_4 有三个实例
- P_1 占有R_2 的一个实例且等待 R_1 的一个实例
- P_2 占有R_1 的一个实例, R_2 的一个实例, 且等待 R_3 的一个实例
- T_3 占有R_3 的一个实例

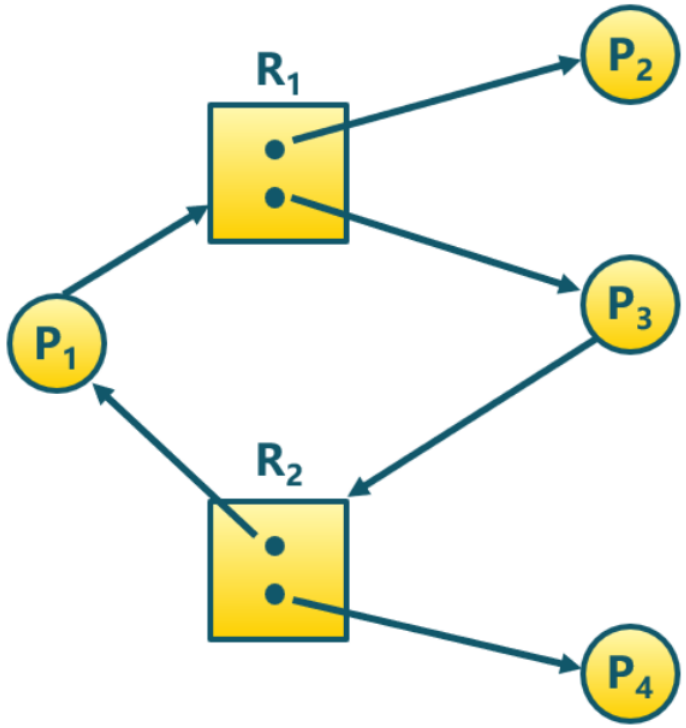


带环的资源分配图

- 可以证明：如果资源分配图没有环，那么系统就没有进程死锁。如果分配图有环，每类资源仅有一个实例，则存在死锁，若每类资源有多个实例，则可能存在死锁。



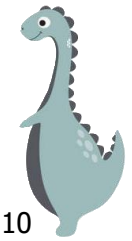
存在死锁的资源分配图



带环且无死锁的资源分配图

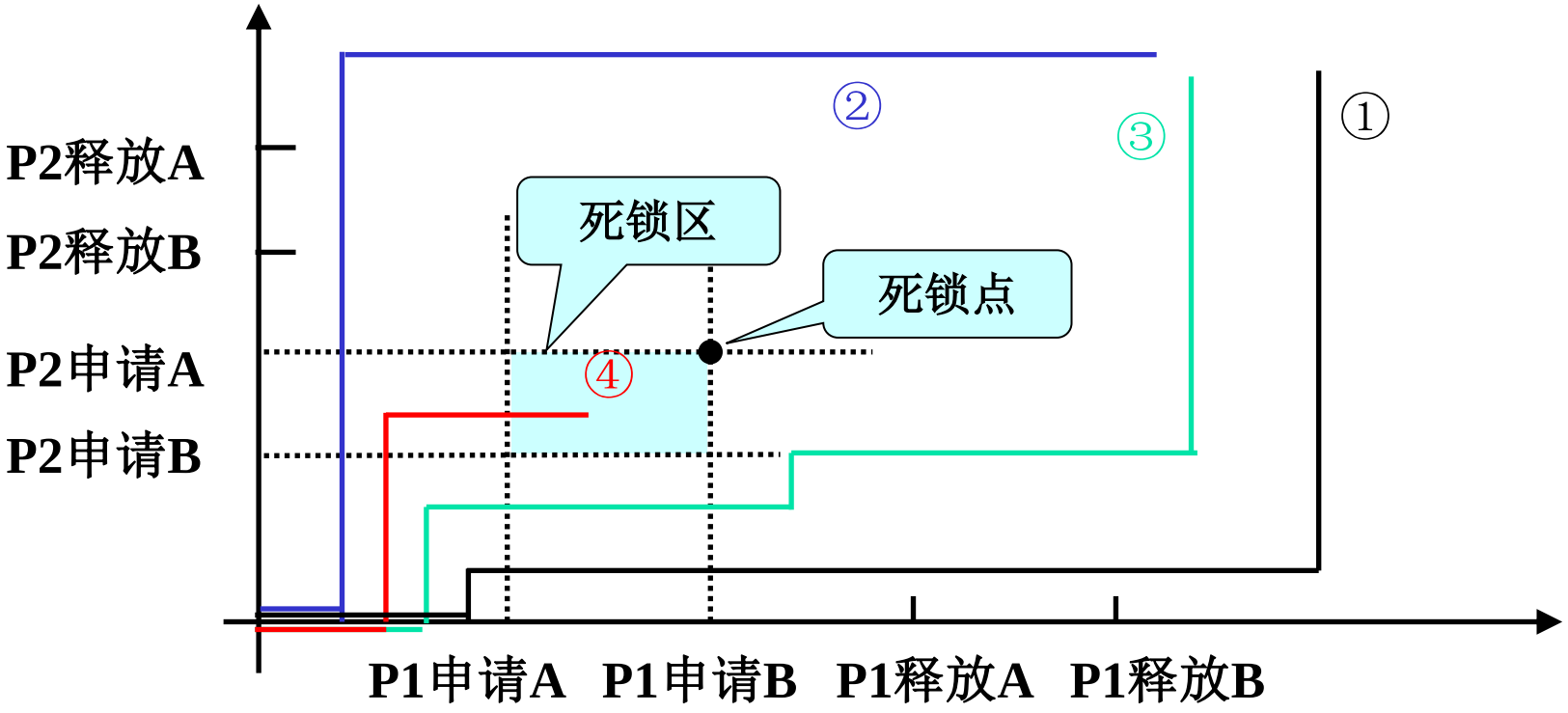
死锁产生的原因

- 死锁产生的原因与资源的使用情况相关
  - **竞争资源**：多个进程竞争资源，而资源又不能同时满足其需求。
    - 可剥夺资源：某进程获得这类资源后，该资源可以被其他进程或系统剥夺。如CPU，存储器。竞争可剥夺资源**不会产生死锁**。
    - 不可剥夺资源：系统将这类资源分配给进程后，再不能强行收回，只能在进程使用完后主动释放。如打印机、读卡机。
  - **进程推进顺序不当**：进程申请资源和释放资源的顺序不当。



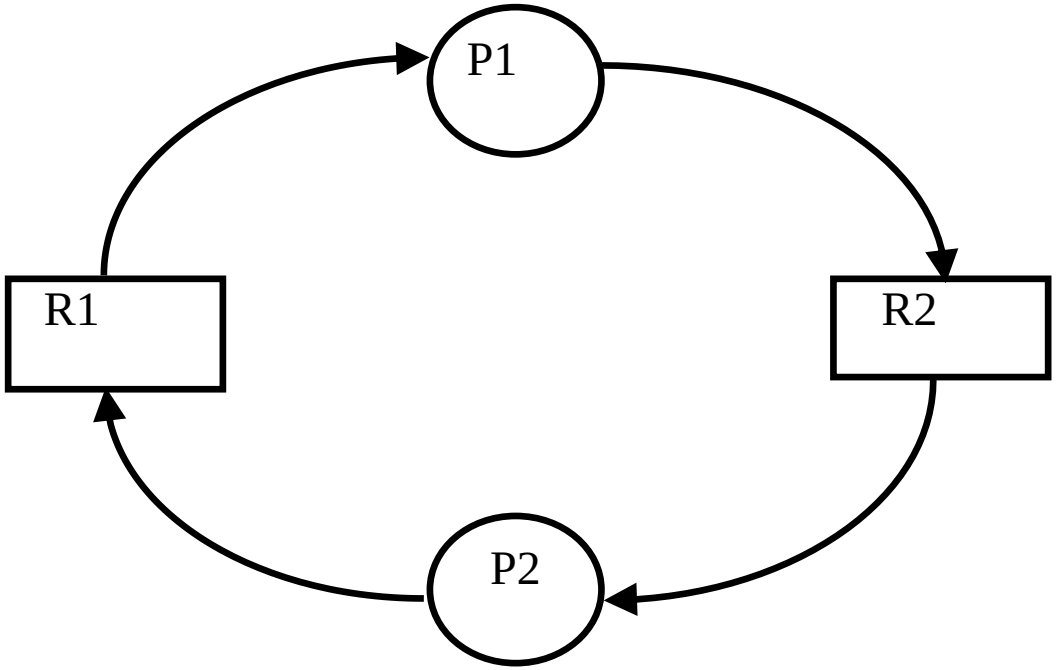
进程推进顺序不当引起的死锁

- 当进程P1、P2共享资源A、B时，若推进顺序合法则不会产生死锁，否则会产生死锁。
- 合法的推进路线：①②③　不合法的推进线路：④



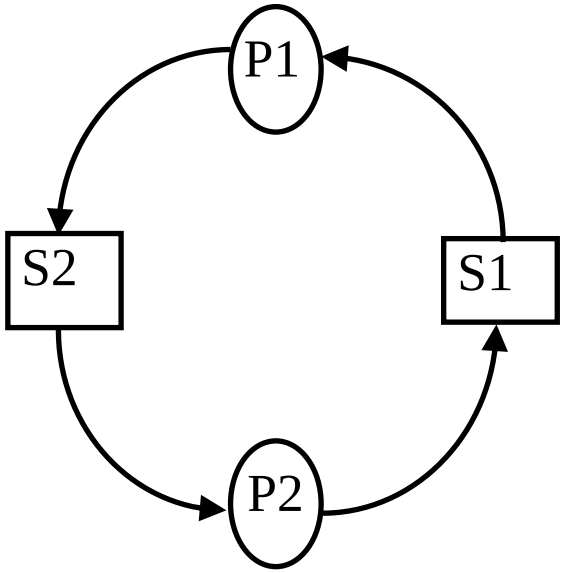
竞争非剥夺资源引起的死锁

- 竞争非剥夺资源例。如，打印机R1和读卡机R2供进程P1和P2共享。



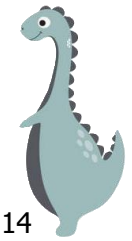
竞争消耗性资源引起的死锁

- 如消息通信按下述顺序进行，则不会发生死锁：
  - P1：...Release(S1)；Request(S2)；...
  - P2：...Release(S2)；Request(S1)；...
- 若按下述顺序，则可能发生死锁：
  - P1：... Request(S2)；Release(S1)；...
  - P2：... Request(S1)；Release(S2)；...



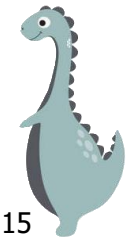
死锁产生的必要条件

- **互斥访问**：在一段时间内某资源仅为一个进程所占有。
- **持有并等待**：又称部分分配条件。当进程因请求资源被阻塞时，已分配资源保持不放。
- **资源非抢占**：进程所获得的资源在未使用完毕之前，不能被其他进程强行夺走。
- **循环等待**：死锁发生时，存在一个进程资源的循环。



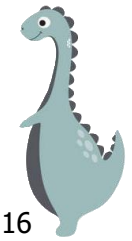
处理死锁的方法

- 确保系统从不进入死锁状态
  - 死锁预防
  - 避免死锁
- 允许系统进入死锁状态，然后检测它并恢复系统
- 忽略死锁，假装死锁在系统中从未发生过
  - 由此带来的问题是什么？



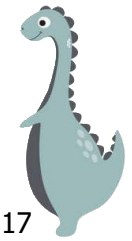
处理死锁的基本方法

- 用于处理死锁的方法主要有：
  - **忽略死锁**。这种处理方式又称鸵鸟算法，指像鸵鸟一样对死锁视而不见。
  - **预防死锁**：设置某些限制条件，通过破坏死锁产生的四个必要条件之一来预防死锁。
  - **避免死锁**：在资源的动态分配过程中，用某种方法来防止系统进入不安全状态。
  - **检测死锁及解除**：系统定期检测是否出现死锁，若出现则解除死锁。



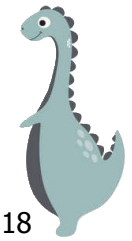
死锁的预防

- 破坏产生死锁的四个必要条件之一来防止发生死锁
  - 互斥条件：是资源本身的特性，无法破坏
  - 占有并等待：必须保证进程申请资源的时候没有占有其他资源
    - 要求进程在执行前一次申请全部的资源——静态资源分配法
    - 或只有没有占有资源时才可以分配资源
    - 资源利用率低，可能出现饥饿



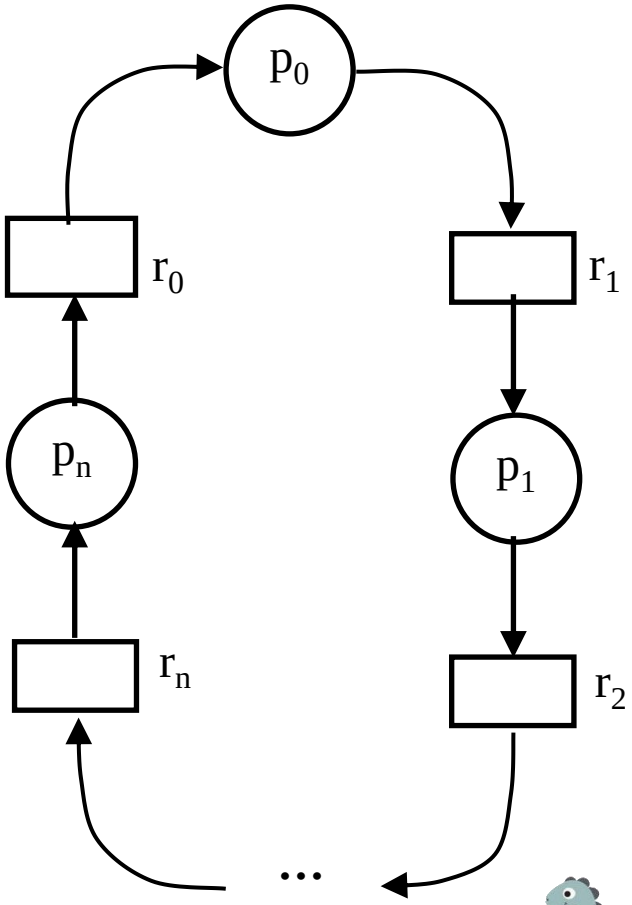
死锁的预防

- 非抢占
  - 如果一个进程占有资源并申请另一个不能立即分配的资源，那么其现已分配的资源都可被抢占。
  - 抢占的资源被添加到该进程正在等待的资源列表中。
  - 只有当进程能够重新获得其旧资源以及其请求的新资源时，才会重新启动该进程。
- 循环等待
  - 对所有资源类型进行完全排序，且要求进程按照递增顺序来申请资源。



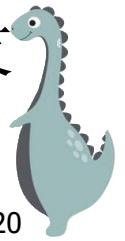
为什么有序资源分配法可以防止死锁

- 假设循环已经出现并且含于环中的进程是p_0、…、p_n，
- 这意味着p_i正占有r_i类资源，而请求r_{i+1}类资源，
- 设函数f能获得资源序号，则有$f(r_i) < f(r_{i+1})$，
- 故$f(r_0) < f(r_1) < \ldots < f(r_n) < f(r_0)$。
- 矛盾，原假设不成立。



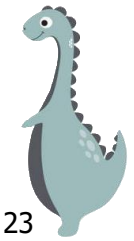
为什么有序资源分配法可以防止死锁

- 采用有序资源分配法，系统中的进程必须按照资源编号的升序申请资源。
- 因此在任一时刻，系统中总会存在一个进程，它占有已申请资源中编号最高的资源，且它继续请求的资源必定是空闲的，因而它可以一直向前推进直至完成。
- 当该进程运行完成后，即会释放它所占有的全部资源。这样剩余进程集合中又会存在一个进程，它占有已申请资源中编号最高的资源，且它继续请求的资源必定是空闲的，因而它也可以一直向前推进直至完成。
- 以此类推，最终所有进程均可运行完成，故不会发生死锁。



死锁的避免

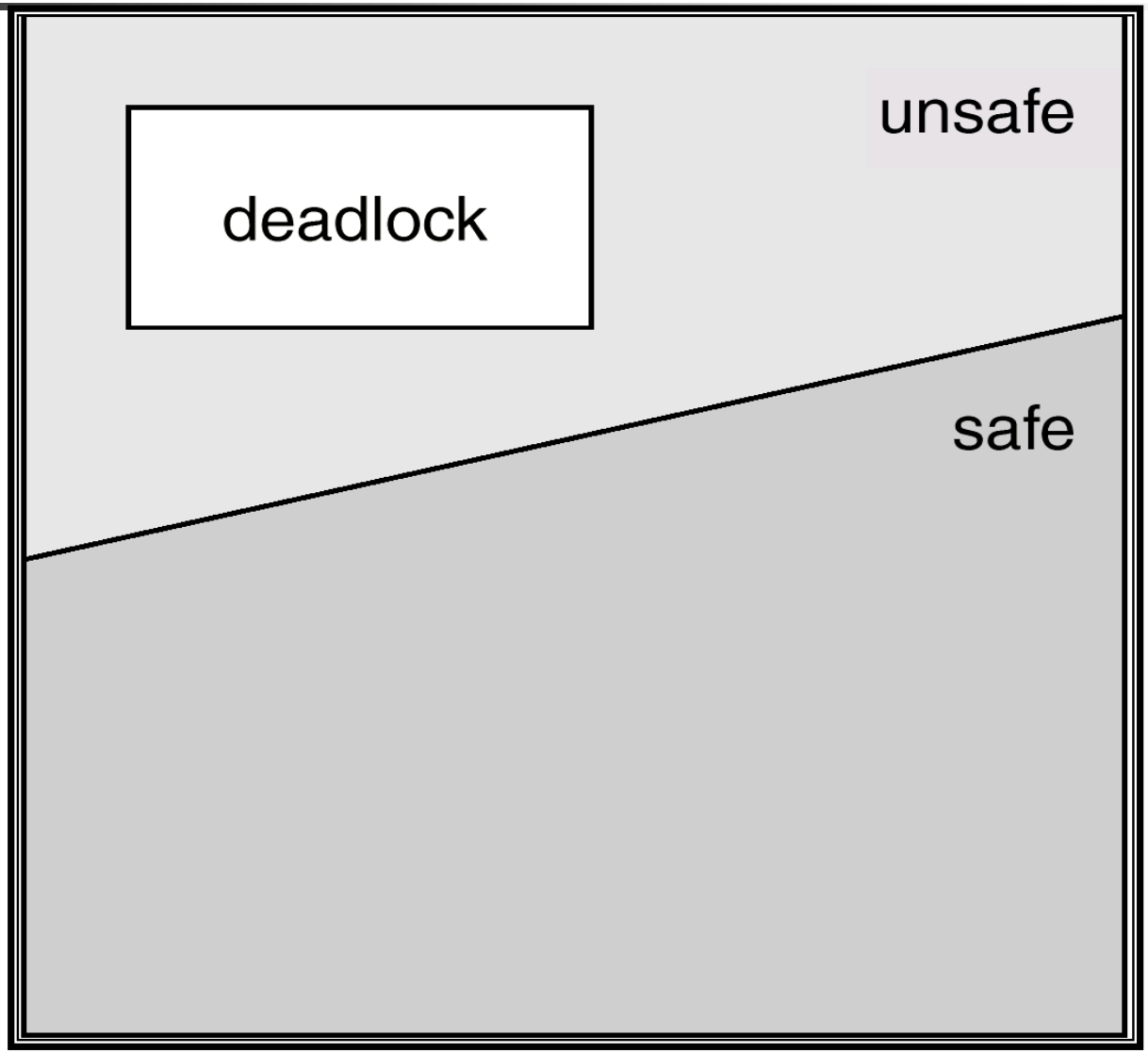
- 死锁的避免是在资源的动态分配过程中，用某种方法防止系统进入不安全状态，从而避免死锁的发生。
- 特点：以较弱的限制获得较高的利用率，但实现有一定难度。



安全状态

- **安全状态**是指系统能按某种顺序如< P_1、P_2... 、P_n>来为每个进程分配其所需的资源，直至最大需求，使每个进程都可以顺利完成，则称此时的系统状态为安全状态，称序列< P_1、P_2、...、P_n >为安全序列。
- 若不存在一个安全序列，则称系统状态为不安全状态。
- 安全状态不是死锁状态，死锁状态是不安全状态

安全、不安全、死锁状态空间



安全状态例

- 假设在T_0时刻，有3 个资源可获得.

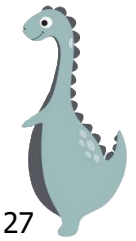
	Max Needs	Current Needs	free
P0	10	5	3
P1	4	2	
P2	9	7	

这时P1可立即得到需要的所有资源，并能将资源归还。

安全状态例

	Max Needs	Current Needs	free
P0	10	5	5
P1	4	2	
P2	9	7	

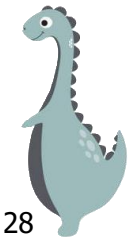
- 接着P0可得到需要的所有资源，并能将资源归还。



安全状态例

	Max Needs	Current Needs	free
P0	10	5	10
P1	4	2	
P2	9	7	

- 最后P2可得到需要的所有资源，并能将资源归还。
- 在T0时刻，系统状态安全。序列<P1、P0、P2>满足安全条件。



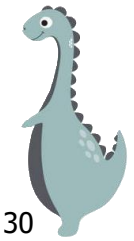
从安全状态转换为不安全状态

- 假设在T1时刻，进程P2申请并得到了1个资源，系统就不安全了。

	Max Needs	Current Needs	free
P0	10	5	2
P1	4	2	
P2	9	6	

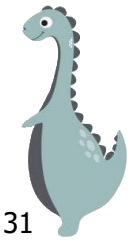
银行家算法

- 最具代表性的死锁避免算法是Dijkstra的银行家算法。
- 每一个进程必须事先声明资源最大使用量
- 当用户申请一组资源时，系统必须确定这些资源的分配是否仍会使系统处于安全状态，如果是就可分配资源，否则等待。



银行家算法的数据结构

- 设n为进程的数目，m为资源类型的数目
- 可用资源向量Available：长度为m的向量。表示每种资源的现有实例数。
  - 如果available[j]=k, 那么R_j类资源现在有k个实例。
- 最大需求矩阵Max：n×m的矩阵，定义了每个进程的最大需求。
  - 如果Max[i,j]=k,那么进程P_i最多可以请求R_j类资源的k个实例



银行家算法的数据结构

- 分配矩阵Allocation： n×m 的矩阵。定义每个进程现在所分配的各类资源实例数目。
  - 如果Allocation[i,j]=k,那么进程P_i当前分配了R_j类资源的k个实例。
  - Allocationi表示进程P_i的分配向量，由矩阵Allocation的第i行构成。
- 需求矩阵Need：n×m 的矩阵，表示每个进程还需要的资源数目。
  - 如果Need(i，j)＝k，表示进程P_i还需R_j类资源k个。
  - $Need_i$表示进程P_i的需求向量，由矩阵Need的第i行构成。
  - Need [i,j] = Max[i,j] – Allocation [i,j].

需求矩阵

- Need：n×m 的矩阵，表示每个进程还需要的资源数目。

  Need: An n×m matrix indicates the remaining resource need of each process.

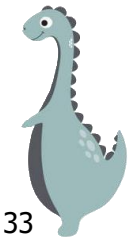
- 如果Need(i，j)＝k，表示进程P_i还需R_j类资源k个。

  If Need[i,j] = k, then P_i may need k more instances of R_j to complete its task.

- $Need_i$表示进程P_i的需求向量，由矩阵Need的第i行构成。

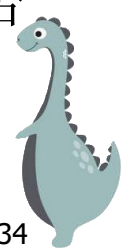
- Note that:

  Need [i,j] = Max[i,j] – Allocation [i,j].



安全性算法 Safety Algorithm

- 1）设Work和Finish分别是长度为m和n的向量。初始化为：
  - Work=available
  - Finish[i]=false　i=1,2, … ,n
- 2）查找这样的进程i使其满足：
  - Finish(i)= false ；
  - $Need_i \leq Work$ ；
  - 如果没有这样的i存在，则进入步骤4)
- 3）当进程Pi获得资源后，可顺利执行直到完成，并释放出分配给它的资源，故应执行：
  - $Work=Work+Allocation_i$
  - Finish[i]=true
  - 转步骤2)
- 4）如果对所有i，Finish[i]=true，那么系统处于安全状态，否则处于不安全状态



资源请求算法（银行家算法）

- 设$Request_i$是进程P_i的请求向量，如$Request_i(j)=k$，那么进程P_i需要R_j类资源k个。
- 当P_i发出资源请求后，系统按下述步骤进行检查
  - 1）如果$Request_i \le Need_i$，则转向步骤2；否则出错。
  - 2）如果$Request_i \le Available$，则转向步骤3；否则P_i等待。
  - 3）试分配并修改数据结构：假设分配P_i请求的资源
    - **$Available = Available - Request_i$；**
    - **$Allocation_i = Allocation_i + Request_i$；**
    - **$Need_i = Need_i - Request_i$；**
  - 4）系统执行安全性算法，检查此次资源分配是否安全。若安全，才正式分配；否则，试分配作废，让进程P_i等待。

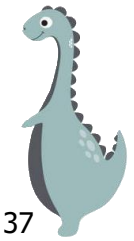
银行家算法例

- 假定系统中有5个进程 P0、P1、P2、P3、P4和三种类型的资源A、B、C，数量分别为12、5、9，在T0时刻的资源分配情况如下所示。

资源情况 / 进程	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	8	5	3	1	1	0	7	4	3	3	3	2
P_1	3	2	3	2	0	1	1	2	2			
P_2	9	0	3	3	0	3	6	0	0			
P_3	2	2	2	2	1	1	0	1	1			
P_4	5	3	3	1	0	2	4	3	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，可得如下所示的T_0时刻的安全性分析。



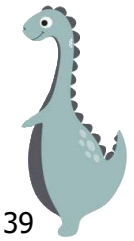
T_0时刻的安全性检查

Need0：7,4,3　Need1：1,2,2　Need2：6,0,0　Need3：0,1,1　Need4：4,3,1
Alloc0：1,1,0　Alloc1：2,0,1　Alloc2：3,0,3　Alloc3：2,1,1　Alloc4：1,0,2
Avail 332

资源情况 / 进程	Work	Need	Alloc	Work+Alloc	Finish
	A B C	A B C	A B C	A B C	
P_1	3 3 2	1 2 2	2 0 1	5 3 3	true
P_3	5 3 3	0 1 1	2 1 1	7 4 4	true
P_4	7 4 4	4 3 1	1 0 2	8 4 6	true
P_2	8 4 6	6 0 0	3 0 3	11 4 9	true
P_0	11 4 9	7 4 3	1 1 0	12 5 9	true

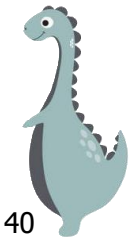
T_0时刻是安全的

- 从上述分析得知，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。



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：
  - 1）$Request_1$(1，0，2) ≤$Need_1$(1，2，2)
  - 2）$Request_1$(1，0，2) ≤ Available(3，3，2)
  - 3）系统先假定可为P_1分配资源，并修改Available、$Allocation_1$、$Need_1$向量，由此形成的资源变化情况如下所示。



为P_1试分配资源后

资源情况 / 进程	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	8	5	3	1	1	0	7	4	3	2	3	0
P_1	3	2	3	3	0	3	0	2	0			
P_2	9	0	3	3	0	3	6	0	0			
P_3	2	2	2	2	1	1	0	1	1			
P_4	5	3	3	1	0	2	4	3	1			

- 4）再利用安全性算法检查此时系统是否安全，可得如下所示的安全性分析。

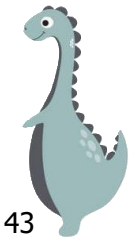
P_1申请资源后的安全性检查

Need0：7,4,3　Need1：0,2,0　Need2：6,0,0　Need3：0,1,1　Need4：4,3,1
Alloc0：1,1,0　Alloc1：3,0,3　Alloc2：3,0,3　Alloc3：2,1,1　Alloc4：1,0,2
Avail 230

资源情况 / 进程	Work	Need	Alloc	Work+Alloc	Finish
	A B C	A B C	A B C	A B C	
P_1	2 3 0	0 2 0	3 0 3	5 3 3	true
P_3	5 3 3	0 1 1	2 1 1	7 4 4	true
P_4	7 4 4	4 3 1	1 0 2	8 4 6	true
P_2	8 4 6	6 0 0	3 0 3	11 4 9	true
P_0	11 4 9	7 4 3	1 1 0	12 5 9	true

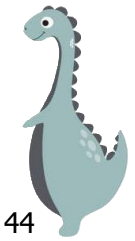
可以为P_1分配资源

- 从上述分析得知，可以找到安全序列<P_1、P_3、P_4、P_0、P_2>，系统安全，可以分配。



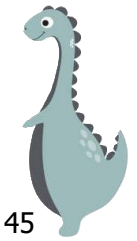
P_4请求资源

- 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) ≤$Need_4$(4，3，1)
  - 2）$Request_4$(3，3，0)＞ Available(2，3，0)，让P4等待。



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）系统先假定可为P_0分配资源，并修改有关数据，如下所示。



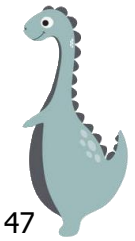
为P_0试分配资源后

资源情况 / 进程	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	8	5	3	1	3	0	7	2	3	2	1	0
P1	3	2	3	3	0	3	0	2	0			
P2	9	0	3	3	0	3	6	0	0			
P3	2	2	2	2	1	1	0	1	1			
P4	5	3	3	1	0	2	4	3	1			

- 4）再利用安全性算法检查此时系统是否安全。从上表中可以看出，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。

死锁检测

- 通过系统的检测机构及时地检测出死锁的发生，然后采取某种措施解除死锁。
- 特点：死锁检测和解除可使系统获得较高的利用率，但是实现难度最大。

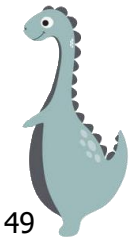


死锁判定法则

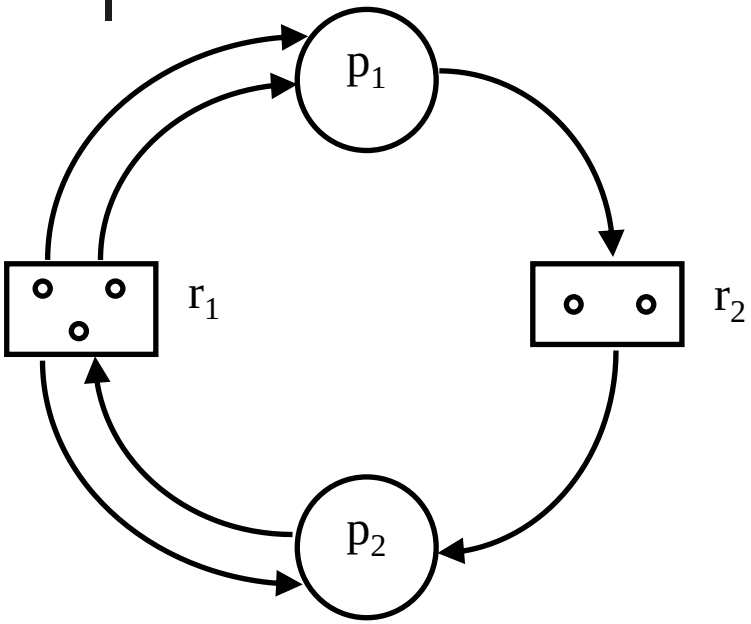
- 将资源分配图简化可以检测系统状态S是否为死锁状态，方法如下：
  - 在资源分配图中，找出一个既不阻塞又非孤立的进程结点p_i，进程p_i获得了它所需要的全部资源，能运行完成然后释放所有资源。这相当于消去p_i的所有请求边和分配边，使之成为孤立结点。
  - 进程p_i释放资源后，可以唤醒因等待这些资源而阻塞的进程，从而可能使原来阻塞的进程变为非阻塞进程。
  - 在进行一系列化简后，若能消去图中所有的边，使所有进程都成为孤立结点，则称该图是可完全简化的；若不能使该图完全化简，则称该图是不可完全简化的。

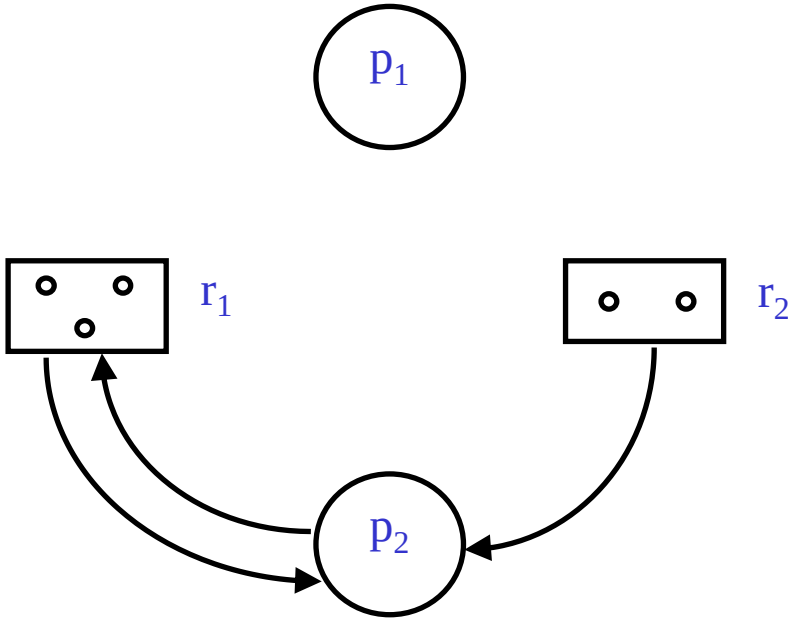
死锁定理

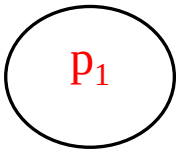
- 可以证明：所有的简化顺序将得到相同的不可简化图。
- S为死锁状态的条件是当且仅当S状态的资源分配图是不可完全简化的。该条件称为死锁定理。

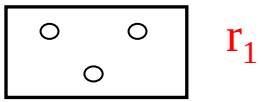


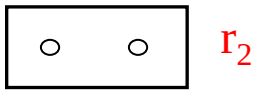
资源分配图简化例

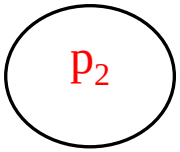






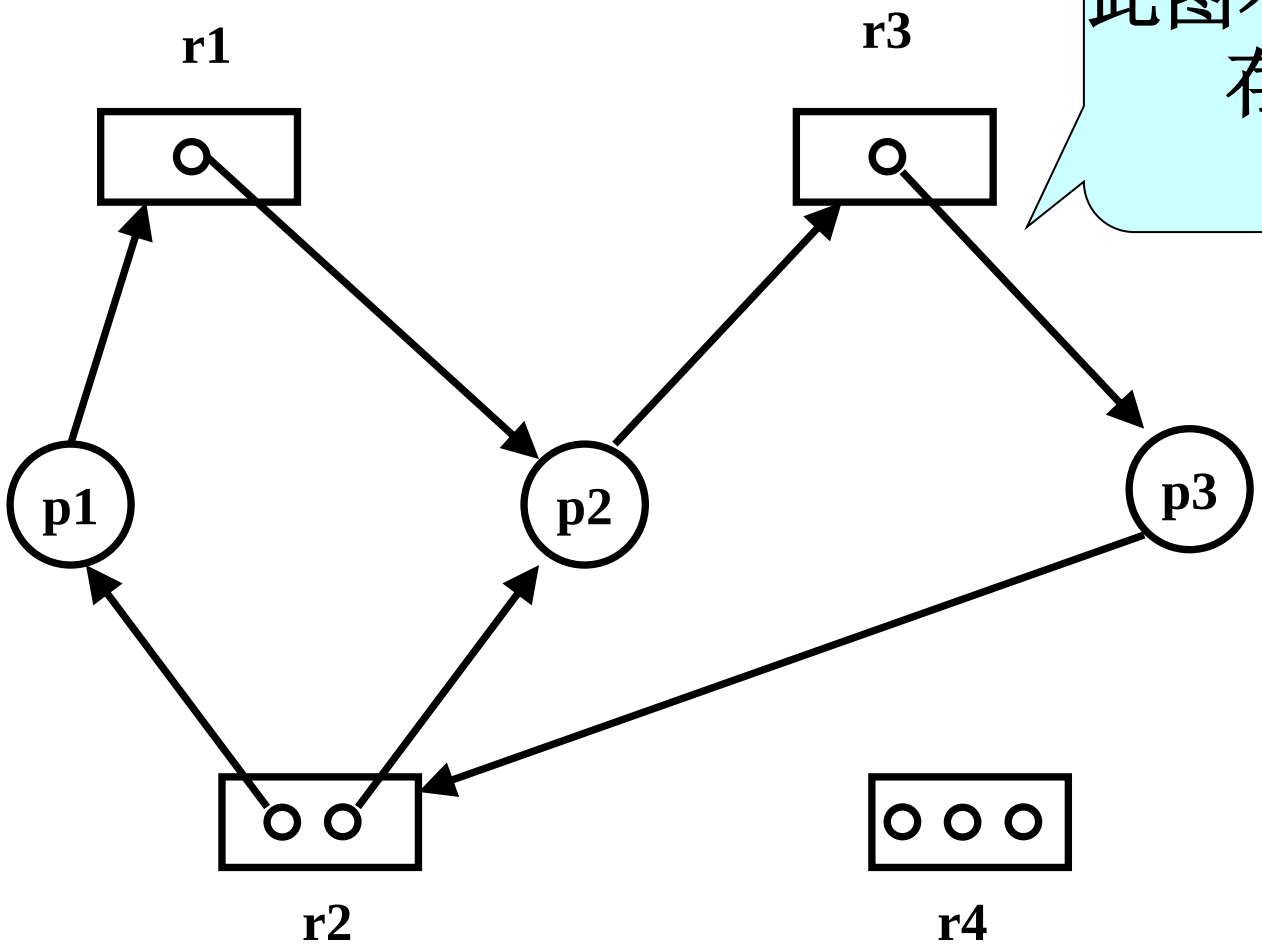






资源分配图简化例

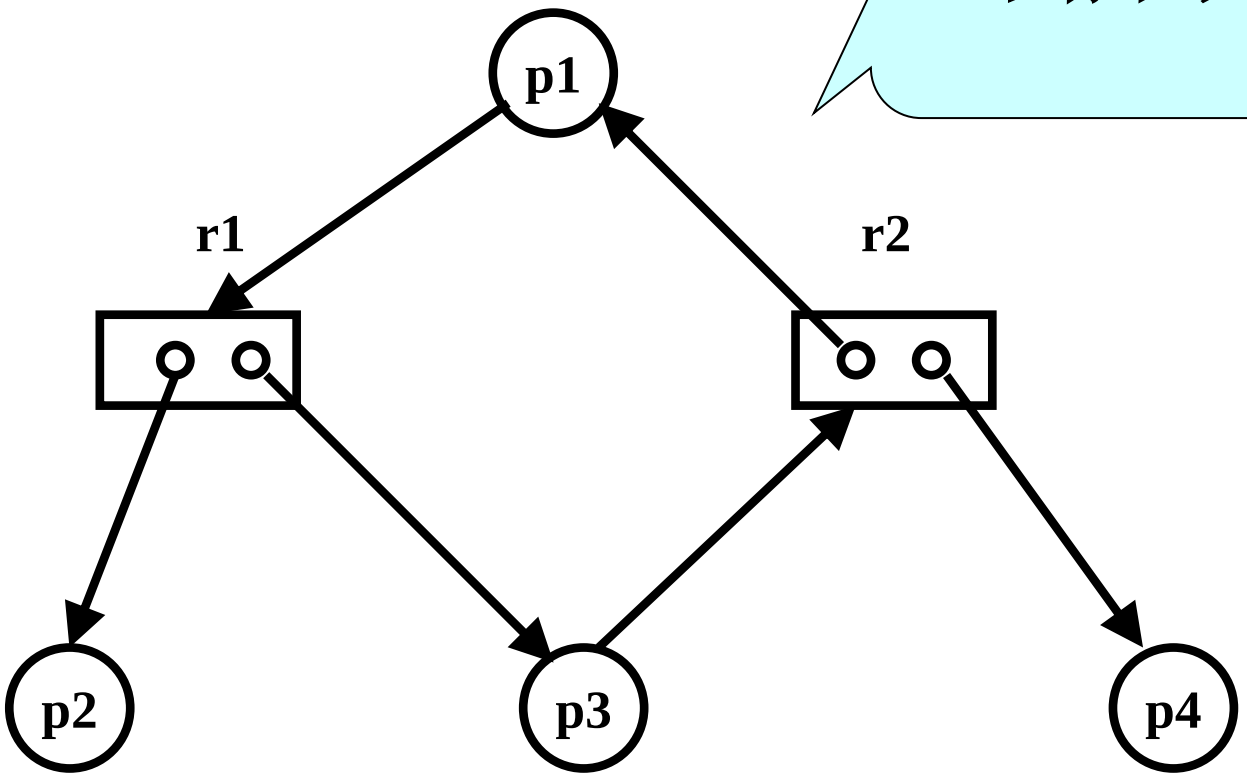
- 下图是否存在死锁？



此图不可完全简化，存在死锁。

资源分配图简化例

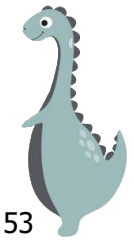
- 下图是否存在死锁？



此图可以完全简化，不存在死锁。

死锁检测算法中的数据结构

- Available：长度为m的向量，表示每类资源的可用数目。
- Allocation： n×m矩阵，表示当前每个进程已分配的资源数目。
- 请求矩阵Request： n×m矩阵，表示当前每个进程的资源请求数目。
- 工作向量Work：表示系统当前可提供资源数。
- 进程集合L：记录当前已不占用资源的进程。



死锁检测的算法

Work=Available;

$L=<L_i \mid Allocation_i=0 \cap Request_i=0>$

for all $L_i \notin L$ do

{

if（$Request_i \leq Work$）

{

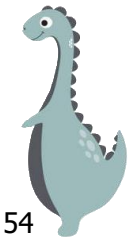
$Work=Work+Allocation_i$;

$L=L \cup L_i$;

}

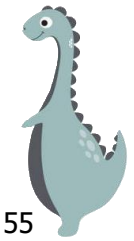
}

$deadlock=\neg(L==<p_1,p_2,\ldots p_n>)$



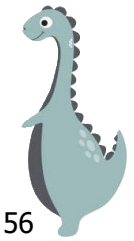
死锁恢复（死锁解除）

- 一旦检测出系统中出现了死锁，就应将陷入死锁的进程从死锁状态中解脱出来，常用的死锁解除方法有两种：
  - 撤消进程法
  - 资源剥夺法



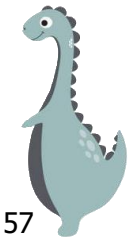
进程终止（撤消进程）

- 通过终止进程消除死锁有两种方法：
  - 终止所有死锁进程：这种方法打破了死锁环，但代价大
  - 一次只终止一个进程直到消除死锁：这种方法开销大，因为每终止一个进程就要检测死锁
- 影响中止进程选择的因素
  - 进程的优先级
  - 进程已经计算了多久，还需要多长时间完成
  - 进程使用的资源，进程完成还需要多少资源
  - 多少个进程需要被终止
  - 进程是交互的还是批处理



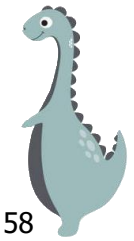
资源抢占（资源剥夺）

- 逐步从进程中抢占资源给其他进程使用，直到死锁环被打破为止。
- 使用抢占来处理死锁，有三个问题需要处理：
  - 选择一个牺牲：最小代价
  - 回退：返回到安全状态，然后重新启动进程
  - 饥饿：同一进程可能总是被选中



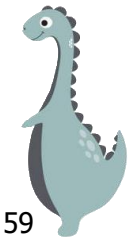
处理死锁的综合方法

- 单独使用处理死锁的某种方法不能全面解决OS中遇到的所有死锁问题。
- 综合解决的办法是：将系统中的资源按层次分为若干类，对每一类资源使用最适合它的办法解决死锁问题。即使发生死锁，一个死锁环也只包含某一层次的资源，因此整个系统不会受控于死锁。



综合方法例

- 将系统的资源分为四个层次：
  - 内部资源：由系统本身使用，如PCB，采用有序资源分配法。
  - 主存资源：采用资源剥夺法。
  - 作业资源：可分配的设备和文件。采用死锁避免法。
  - 交换空间：采用静态分配法。



练习题

- 8.3 Consider the following snapshot of a system:

	Allocation	Max	Available
	A B C D	A B C D	A B C D
T_0	0 0 1 2	0 0 1 2	1 5 2 0
T_1	1 0 0 0	1 7 5 0	
T_2	1 3 5 4	2 3 5 6	
T_3	0 6 3 2	0 6 5 2	
T_4	0 0 1 4	0 6 5 6	

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix Need?

b. Is the system in a safe state?

c. If a request from thread T1 arrives for (0,4,2,0), can the request be granted immediately

练习题

8.18 Which of the six resource-allocation graphs shown in Figure 8.12 illustrate deadlock? For those situations that are deadlocked, provide the cycle of threads and resources. Where there is not a deadlock situation, illustrate the order in which the threads may complete execution.

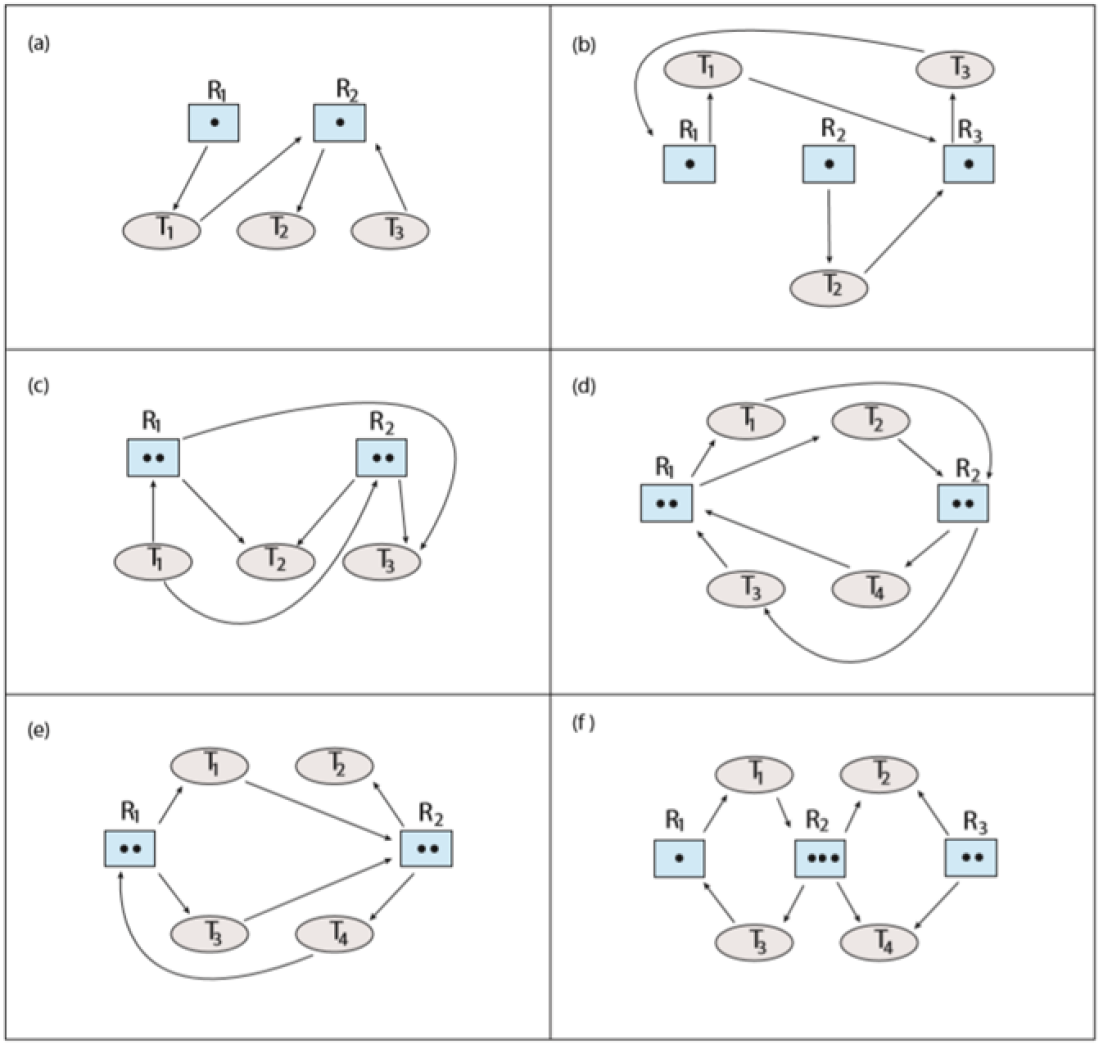


Figure 8.12 Resource-allocation graphs for Exercise 8.18.

选择题1

- 要防止死锁的发生，可以通过破坏这四个必要条件之一来实现，但破坏 _____ 条件是不太实际的。

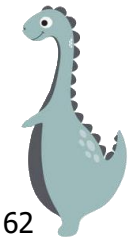
  A.循环等待　　B.部分分配

  C.不可抢占　　D.互斥

- 为多道程序提供的可共享资源不足时，可能出现死锁。但是，不适当的 _____ 也可能产生死锁。

  A.分配队列优先权　　B.进程推进顺序

  C.资源的线性分配　　D.进程优先权



选择题2

- 采用资源剥夺法可以解除死锁，还可以采用 _____ 方法解除死锁。

  A.拒绝分配新资源　　B.修改信号量

  C.执行并行操作　　D.撤消进程

- 在 _____ 的情况下，系统出现死锁。

  A. 计算机系统发生了重大故障

  B. 有多个封锁的进程同时存在

  C. 若干进程因竞争资源而无休止地相互等待他方释放已占有的资源

  D. 资源数大大小于进程数或进程同时申请的资源数大大超过资源总数

选择题3

- 银行家算法在解决死锁问题中是用于 _____ 的。

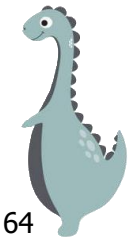
  A.检测死锁　　B.预防死锁

  C.避免死锁　　D. 解除死锁

- 资源的有序分配策略可以破坏 _____ 条件。

  A.占有且等待资源　　B.循环等待资源

  C. 非抢夺资源　　D.互斥使用资源



选择题4

- 某系统中有3个并发进程，都需要同类资源4个，试问该系统不会发生死锁的最少资源数是_____。

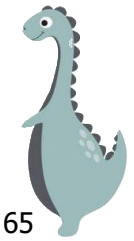
  A. 9　　B. 10

  C. 11　　D. 12

- 有序资源分配方法属于_____方法。

  A．死锁预防　　B．死锁避免

  C．死锁检测　　D．死锁解除



选择题5

- 不能防止死锁的资源分配策略是_____。

  A．剥夺式分配方式　B．按序分配方式

  C．静态分配方式　D．互斥使用分配方式

- 某计算机系统中有6台打印机，多个进程均最多需要2台打印机，规定每个进程一次仅允许申请一台打印机。为保证一定不发生死锁，则允许参与打印机资源竞争的最大进程数是__。

  A．3　B．4　C．5　D．6

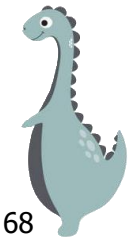
填空题

- 在有m个进程的系统中出现死锁时，死锁进程的个数k应该满足的条件是_____ 。
- 银行家算法中，当一个进程提出的资源请求将导致系统从 ___①___ 进入 ___②___时，系统就拒绝它的资源请求。
- 对待死锁，一般应考虑死锁的预防、避免、检测和解除四个问题。典型的银行家算法是属于 ___①___ ，破坏环路等待条件是属于 ___②___ ，而剥夺资源是 ___③___ 的基本方法。

考研题1

- 某计算机中有8台打印机，由K个进程竞争使用，每个进程最多需要3台打印机。该系统可能会发生死锁的K的最小值为（）。09

  A、2　B、3　C、4　D、5



考研题2

- 某时刻进程的资源使用情况如下表所示：

进 程	已分配资源			尚需资源			可用资源		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	0	0	0	0	1	0	2	1
P2	1	2	0	1	3	2			
P3	0	1	1	1	3	1			
P4	0	0	1	2	0	0			

- 此时的安全序列是____。11

A、P1，P2，P3，P4　　B、P1，P3，P2，P4

C、P1，P4，P3，P2　　D、不存在

考研题3

- 假设**5**个进程**P0**，**P1**，**P2**，**P3**，**P4**共享三类资源**R1**、**R2**、**R3**，这些资源总数分别为**18**，**6**，**22**。**T0**时刻的资源分配情况如下表所示，此时存在一个安全序列是：**12**

	已分配			最大需求		
	R1	**R2**	**R3**	**R1**	**R2**	**R3**
P0	**3**	**2**	**3**	**5**	**5**	**10**
P1	**4**	**0**	**3**	**5**	**3**	**6**
P2	**4**	**0**	**5**	**4**	**0**	**11**
P3	**2**	**0**	**4**	**4**	**2**	**5**
P4	**3**	**1**	**4**	**4**	**2**	**4**

A. P0,P2,P4,P1,P3

B. P1,P0,P3,P4,P2

C. P2,P1,P0,P3,P4

D. P3,P4,P2,P1,P0