華中科技大學
计算机科学与技术学院
School of Computer Science & Technology, HUST

操作系统原理

第五章 资源分配与调度

授课人：郑然

资源分配与调度——主要内容

- 资源管理概述
- 资源分配的机构和策略
- 死锁

1. 资源管理功能

(1) 资源数据结构的描述

包含资源的物理名、逻辑名、类型、地址、分配状态等信息。

(2) 确定资源的分配原则 (调度原则)

决定资源应分给谁，何时分配，分配多少等问题。

(3) 实施资源分配

执行资源分配；资源收回工作。

(4) 存取控制和安全保护

对资源的存取进行控制并对资源实施安全保护措施。

2. 资源的静态分配和动态分配

(1) 资源的静态分配

系统对作业一级采用资源静态分配方法。

系统在调度作业时，根据作业所需资源进行分配；并在作业运行完毕时，收回所分配的全部资源。这种分配通常称为资源的静态分配。

(2) 资源的动态分配

系统对进程一级采用资源动态分配方法。

系统在进程运行中，根据进程提出的资源需求，进行资源的动态分配和回收。这种分配通常称为资源的动态分配。

3. 虚拟资源

(1) 操作系统对资源区分二种不同的概念

- 物理资源 (实资源)
- 虚拟资源 (逻辑资源)

(2) 目的

- 方便用户使用
- 资源可动态分配，提高资源利用率

(3) 计算机系统中的物理资源与虚拟资源分析

资源类别	物理资源	虚拟(逻辑)	映射
处理机	CPU	进程	进程调度
存储器	主存	虚存 (程序地址空间)	地址映射
设备	外部设备	逻辑设备 虚拟设备	设备分配 动态映射
信息	文件物理结构	文件逻辑结构	磁盘空间分配 文件目录查找

资源分配与调度——主要内容

- 资源管理概述
- 资源分配的机构和策略
- 死锁

1. 资源分配的机构

(1) 资源描述器

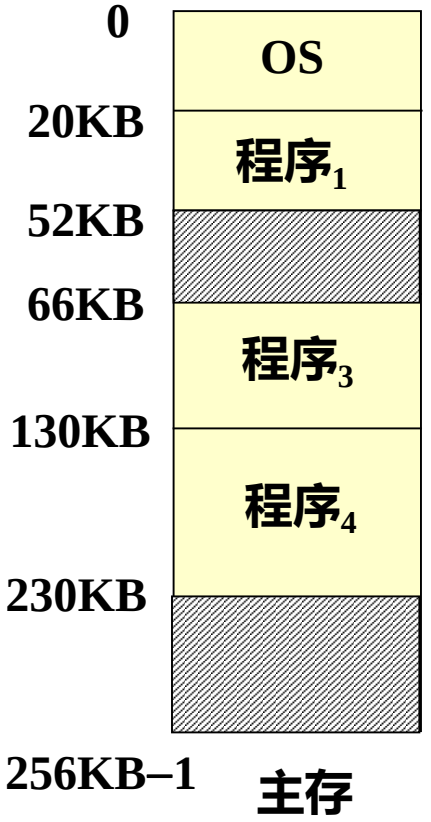
① 资源描述器定义

描述描述各类资源的最小分配单位的数据结构称为资源描述器 rd。

如：主存分区分配方法中，最小分配单位为主存分区。

② 资源描述器内容

资源名、资源类型、最小分配单位的大小、地址、分配标志、描述器链接信息、存取权限、密级、存取时间



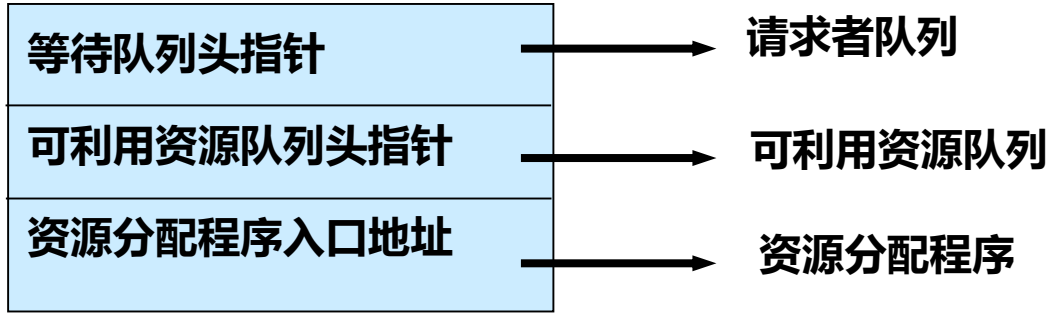
内存分布状况图

(2) 资源信息块

① 资源信息块定义

描述某类资源的请求者、可用资源和该类资源分配程序等必要信息的数据结构。

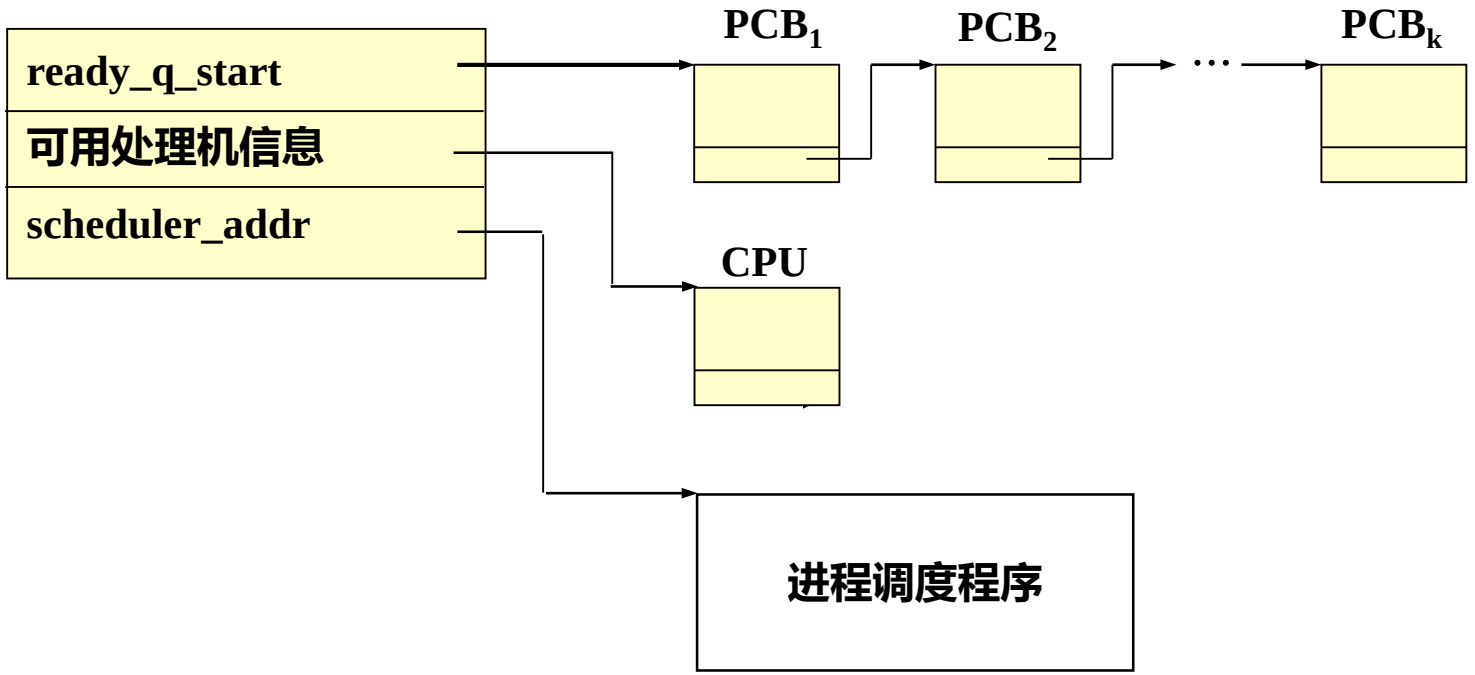
② 资源信息块内容



资源信息块示意图

(3) 资源信息块例

中央处理机资源信息块内容



中央处理机资源信息块示意图

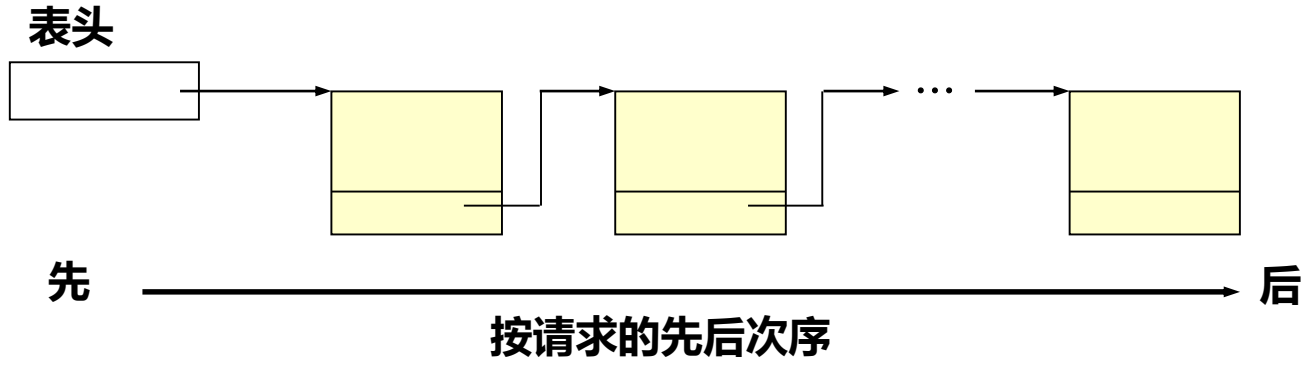
2. 资源分配策略

(1) 常用的资源分配策略

① 先请求先服务

- 每一个新产生的请求均排在队尾；
- 当资源可用时，取队首元素，并满足其需要。

排序原则：按请求的先后次序排序。

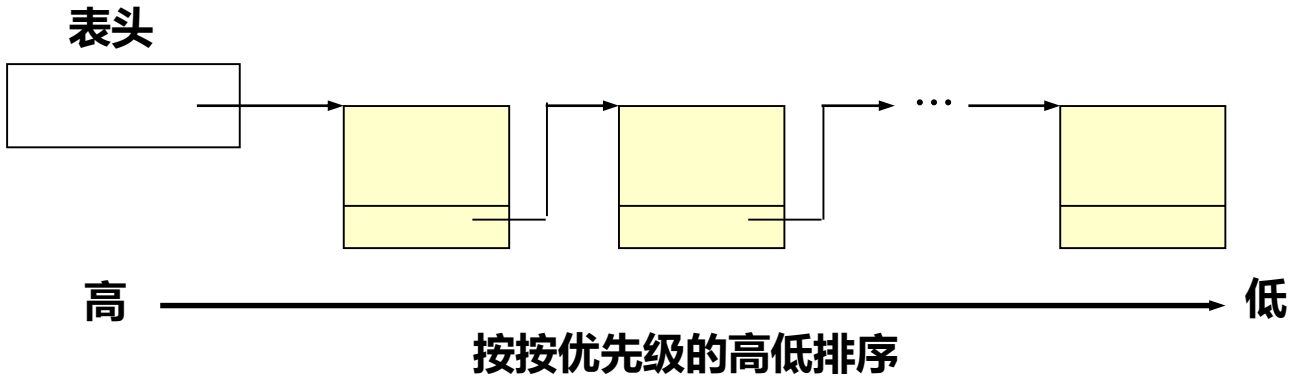


按自然顺序排列的资源请求队列

② 优先调度

- 对每一个进程指定一个优先级；
- 每一个新产生的请求，按其优先级的高低插到相应的位置；
- 当资源可用时，取队首元素，并满足其需要。

排序原则：按优先级的高低排序。



按优先级高低排列的资源请求队列

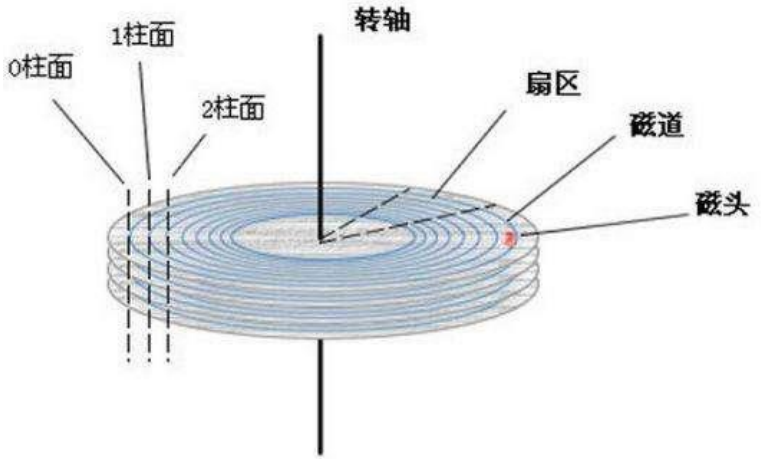
③ 针对设备特性的调度策略

i 调度的目标

当有大量I/O请求时，降低完成这些I/O服务的总时间。

ii **例：**讨论对磁盘访问的调度，有如下5个请求。

柱面号	盘面号	块号
5	2	1
5	3	8
5	3	5
40	6	3
2	7	7



硬盘示意图

每张盘片由若干个磁道和若干个扇区组成
从外向内分别为0磁道、1磁道、2磁道……
不同盘片的同一磁道构成一圆柱面称为柱面
柱面由外向内依次为0柱面、1柱面、2柱面……
磁盘将信息按扇区存入

iii 移臂调度

总是选取与当前移动臂前进方向上最近的那个I/O请求，使移臂距离最短。

◆对磁盘访问的5个请求，按移臂调度应作如下调整。

柱面号	盘面号	块号
5	2	1
5	3	8
5	3	5
40	6	3
2	7	7

柱面号	盘面号	块号
2	7	7
5	2	1
5	3	8
5	3	5
40	6	3

iv 旋转调度

总是选取与当前读写头最近的那个I/O请求，使旋转圈数最少。

◆对磁盘访问的5个请求，再按旋转调度应作如下调整。

柱面号	盘面号	块号
5	2	1
5	3	8
5	3	5
40	6	3
2	7	7

柱面号	盘面号	块号
2	7	7
5	2	1
5	3	5
5	3	8
40	6	3

资源分配与调度——主要内容

- 资源管理概述
- 资源分配的机构和策略
- 死锁

1. 什么是死锁

(1) 死锁的例子

设备共享

进程 p_1、p_2共享一台打印机和一台输入机

时刻 t_1：进程 p_1 —— 占用打印机，

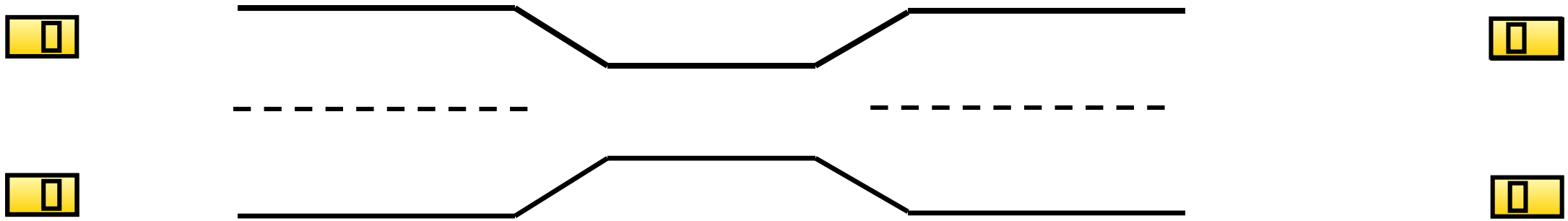
进程 p_2 —— 占用输入机；

时刻 t_2：进程 p_1 —— 又请求输入机，

进程 p_2 —— 又请求打印机。

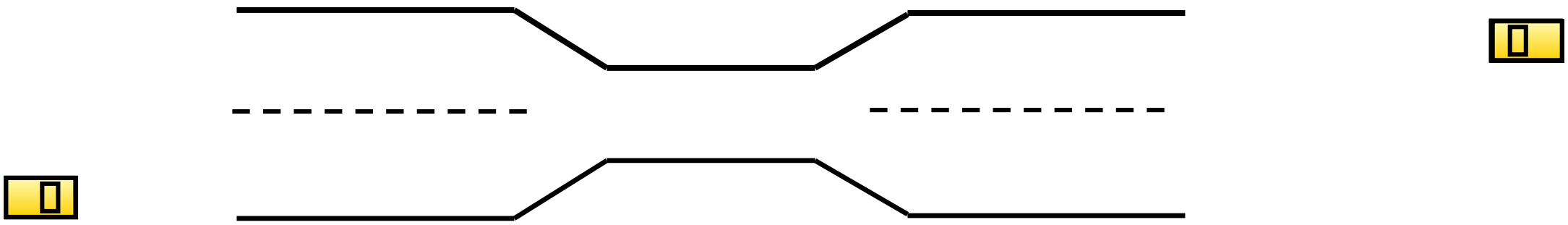
时刻t_2后，系统出现僵持局面，即出现了死锁现象。

死锁示例：单向通行桥梁



- 桥梁只能单向通行
- 桥的每个部分可视为一个资源
- 可能出现死锁
  - 对向行驶车辆在桥上相遇
  - 解决方法：一个方向的车辆倒退(资源抢占和回退)

死锁示例：单向通行桥梁



- 桥梁只能单向通行
- 桥的每个部分可视为一个资源
- 可能出现死锁
- 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

(2) 用信号灯的P、V操作描述死锁

设进程p_1与进程p_2共享一台打印机(r_1) 和一台输入机(r_2)，

用信号灯的p、v操作表示资源的申请和释放。

信号灯设置—— **s_1：表示r_1可用，初值为1**

s_2：表示r_2可用，初值为1

讨论两种资源请求序列，哪种情况可能产生互相死等的局面。

程序描述如下：

资源分配与调度——死锁

程序描述1

进程p_1	进程p_2
⋮	⋮
$p(s_1)$;	$p(s_2)$;
占用r_1	占用r_2
$v(s_1)$;	$v(s_2)$;
⋮	⋮
$p(s_2)$;	$p(s_1)$;
占用r_2	占用r_1
$v(s_2)$;	$v(s_1)$;
⋮	⋮
⋮	⋮

程序描述2

进程p_1	进程p_2
⋮	⋮
$p(s_1)$;	$p(s_2)$;
占用r_1	占用r_2
$p(s_2)$;	$p(s_1)$;
又占用r_2	又占用r_1
⋮	⋮
$v(s_1)$;	$v(s_2)$;
⋮	⋮
$v(s_2)$;	$v(s_1)$;
⋮	⋮

程序描述2，有可能出现死锁。

(3) 什么是死锁

在**两个或多个并发进程**中，如果每个进程**持有某种资源**而**又都等待着别的进程释放它或它们现在保持着的资源**，否则就不能向前推进。此时，称**这一组进程产生了死锁。**

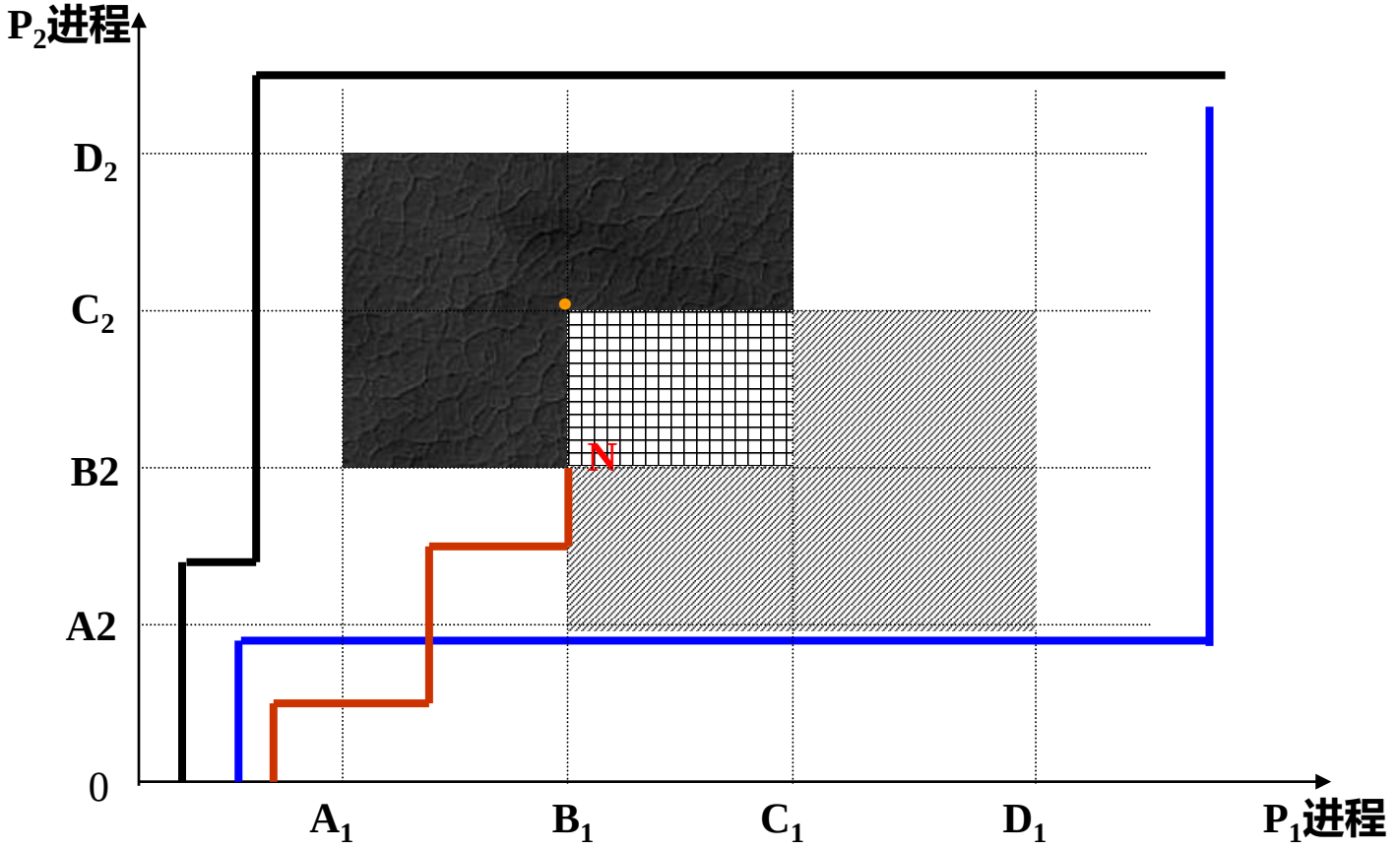
2. 死锁的起因和条件

(1) 引起死锁的原因

① 系统资源不足

② 进程推进顺序非法

(2) 死锁图解



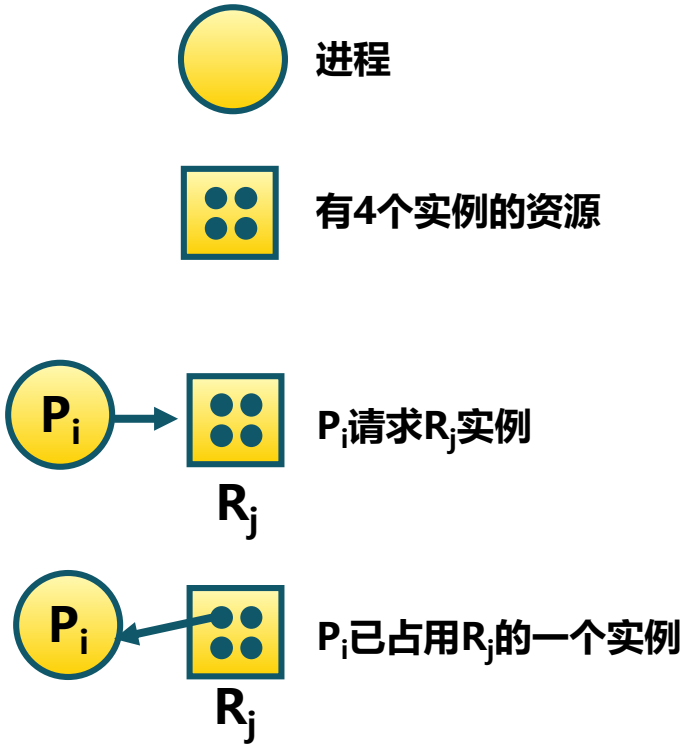
- A1： p1 request (r1)　B1： p1 request (r2)　C1： p1 release (r1)　D1： p1 release (r2)
- A2： p2 request (r2)　B2： p2 request (r1)　C2： p2 release (r2)　D2： p2 release (r1)

(3) 资源分配图

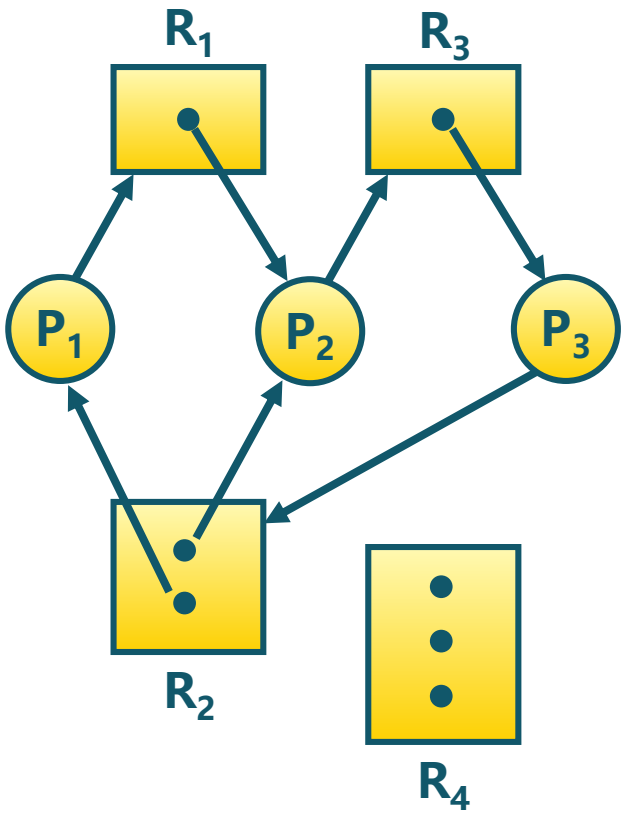
描述资源和进程间的分配和占用关系的有向图

- 两类顶点
  - 系统中的所有进程
    $P = \{P_1, P_2, \ldots, P_n\}$
  - 系统中的所有资源
    $R = \{R_1, R_2, \ldots, R_m\}$

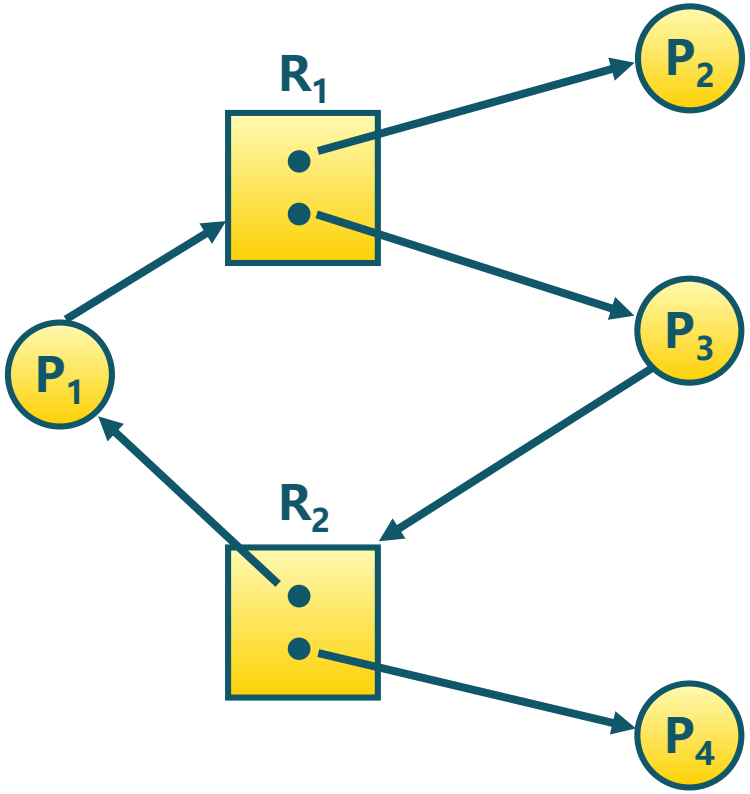
- 两类有向边
  - 资源请求边
    进程P_i请求资源R_j：$P_i \rightarrow R_j$
  - 资源分配边
    资源R_j已分配给进程P_i：$R_j \rightarrow P_i$



资源分配图示例



存在死锁



有循环等待，但没有锁死

如何预防/避免死锁的产生?

(4) 产生死锁的必要条件

① 互斥条件

涉及的资源是非共享的，即为临界资源。

② 不剥夺条件

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。

③ 部分分配

进程每次申请它所需要的一部分资源。在等待一新资源的同时，进程继续占用已分配到的资源。

④ 环路条件

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下一个进程所请求。

死锁的处理策略

预防死锁

- 往往采用静态策略对资源分配进行控制，使得系统永远不可能发生死锁
- 优点是简单，缺点是会导致系统资源利用率降低

避免死锁

- 只在资源分配的时间节点上进行判断，确保改组进程不会发生死锁才进行资源分配
- 优点是能够提高系统资源利用率，缺点是每次资源分配都需要计算

解除死锁

- 允许死锁的发生，操作系统完成死锁的检测和解除

死锁的处理策略——预防死锁

- **核心思想：破坏死锁的四个必要条件之一**
- **破坏互斥条件**
  - 将临界资源转化为共享资源：1. 非独占使用；2. 虚拟化
  - 例如：SPOOLing技术（见第8章）
- **缺点**
  - 需要辅助数据结构和软件来实现资源使用特性的转换
  - 不是所有的临界资源都可以转化为可共享（多进程同时使用）的资源，应用场景有限

死锁的处理策略——预防死锁

破坏不剥夺条件

- 方案一，**改造申请者**：当某个进程请求新的资源得不到满足时，它必须立即释放保持的所有资源，待以后需要时再重新申请。即使某些资源尚未使用完，也需要主动释放，从而破坏了不可剥夺条件。
- 方案二，**改造占用者**：当某个进程需要的资源被其他进程所占有的时候，可以由操作系统协助，将想要的资源强行剥夺。

缺点

- 实现较为复杂，需要操作系统提供大量原语
- 可能导致进程反复申请和释放资源，降低系统资源利用率
- 可能导致**活锁**现象（任务或者执行者没有被阻塞，但由于某些条件没有满足，导致一直重复尝试—失败—尝试—失败的过程）

死锁的处理策略——预防死锁

破坏请求和保持条件

- 资源分配时，采用**静态资源分配法。**
- 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，不让它投入运行。一旦投入运行后，这些资源就一直归它所有，该进程执行过程中不会再请求任何别的资源。

缺点

- 系统资源利用率极低
- 可能导致进程**饥饿**

死锁的处理策略——预防死锁

破坏循环等待条件

- 采用**有序资源分配法。**
- 系统中所有资源都给定一个唯一的编号，所有分配请求必须以上升的次序进行。当遵守上升次序的规则时，若资源可用，则予以分配；否则，请求者等待。

缺点

- 必须按规定次序申请资源，用户编程麻烦。
- 进程实际使用资源的顺序可能和编号递增顺序不一致，会导致资源浪费。

死锁的处理策略——避免死锁

- **每次进行资源分配时进行计算，只有确认不会出现死锁才允许分配**
  - 要求进程声明需要资源的最大数目
  - 限定提供与分配的资源数量，确保满足进程的最大需求
  - 动态检查资源分配状态，确保不会出现环形等待

死锁的处理策略——避免死锁

系统状态分析

(1) 初始状态描述

假定一个系统包括n个进程和m类资源，表示如下

① 一组确定的进程集合，记作：

$$p=\{p_1，p_2，\ldots，p_i，\ldots，p_n\}$$

② 一组不同类型的资源集合，记作：

$$r=\{r_1，r_2，\ldots，r_j，\ldots，r_m\}$$

③ 矢量w说明各类可利用资源的总的数目

$$w=\{w_1，w_2，\ldots，w_j，\ldots，w_m\}$$

死锁的处理策略——避免死锁

(2) 资源请求(demand)矩阵

在时刻 t 资源请求矩阵，表示如右

d_{ij} 表示进程p_i还需要j类资源的数目

$$\mathbf{d(t)} = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1m} \\ d_{21} & d_{22} & \cdots & d_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ d_{n1} & d_{n2} & \cdots & d_{nm} \end{bmatrix}$$

(3) 资源分配(allocation)矩阵

在时刻 t 资源分配矩阵，表示如右

a_{ij} 表示进程p_i已占有j类资源的数目

$$\mathbf{a(t)} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}$$

死锁的处理策略——避免死锁

关于安全状态

A **safe state** is one in which there is at least one sequence of resource allocations to processes that does not result in a deadlock (i.e., all of the processes can be run to completion). An **unsafe state** is, of course, a state that is not safe.

《Operating Systems - Internals and Design Principles》(8th edition) -- William Stallings

系统资源分配的安全状态

当进程请求某类资源时，进程对该类资源的需求量小于当前时刻系统所拥有的该类资源的数目，那么满足进程的这次请求，系统是安全的。

- 系统处于不安全状态，可能出现死锁
  避免死锁就是确保系统不会进入不安全状态
- 系统处于安全状态，一定没有死锁

死锁的处理策略——避免死锁

P个进程共享m个同类资源，每一个资源在任一时刻只能供一个进程使用，每一进程对任一资源都只能使用一有限时间，使用完便立即释放。并且每个进程对该类资源的最大需求量小于该类资源的数目。设所有进程对资源的最大需要数目之和小于$p+m$。试证：在该系统中不会发生死锁。

银行家算法

- 以银行借贷分配策略为基础，判断并保证系统处于安全状态
  - 客户在第一次申请贷款时，声明所需最大资金量，在满足所有贷款要求并完成项目时，及时归还
  - 在客户贷款数量不超过银行拥有的最大值时，银行家尽量满足客户需要
  - 类比

    银行家 ⟷ 操作系统
    资金 ⟷ 资源
    客户 ⟷ 申请资源的进程

申请者事先说明对各类资源的最大需求量。在进程活动期间动态申请某类资源时，由系统审查：若分配，系统是否仍然处于安全状态？若是，则予以分配；若否，则拒绝分配。

银行家算法

申请者事先说明对各类资源的最大需求量。在进程活动期间动态申请某类资源时，由系统审查：若分配，系统是否仍然处于安全状态？若是，则予以分配；若否，则拒绝分配。

例：系统拥有某类资源10个，现有进程P、Q、R共享该类资源，它们申请该类资源的最大需求量如下。

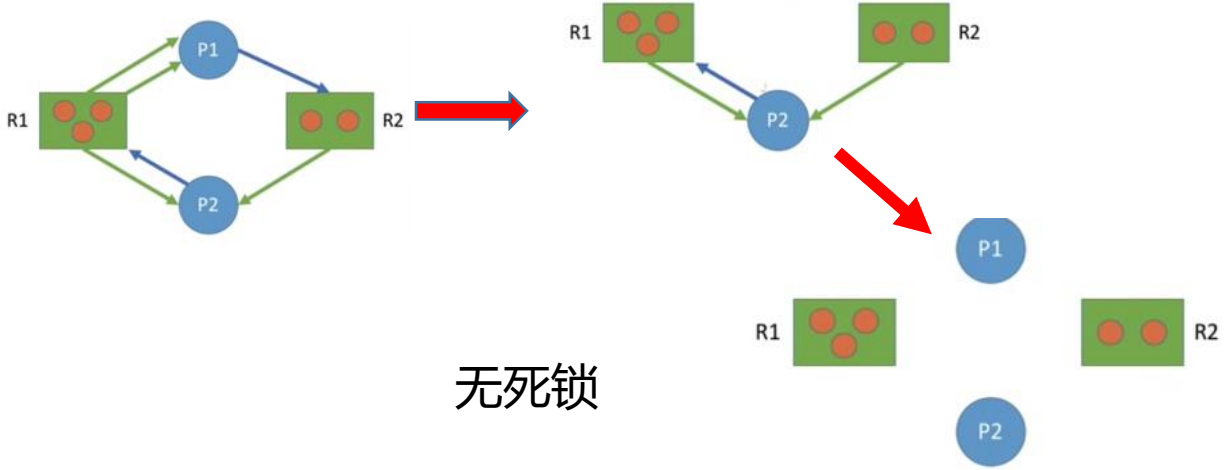
进程	最大需求量	已占有资源	现申请资源个数
P	8	4	**1**
Q	4	2	**1**
R	9	2	**1**

当这些进程动态申请资源时，按银行家算法应如何分配？能保证不发生死锁。

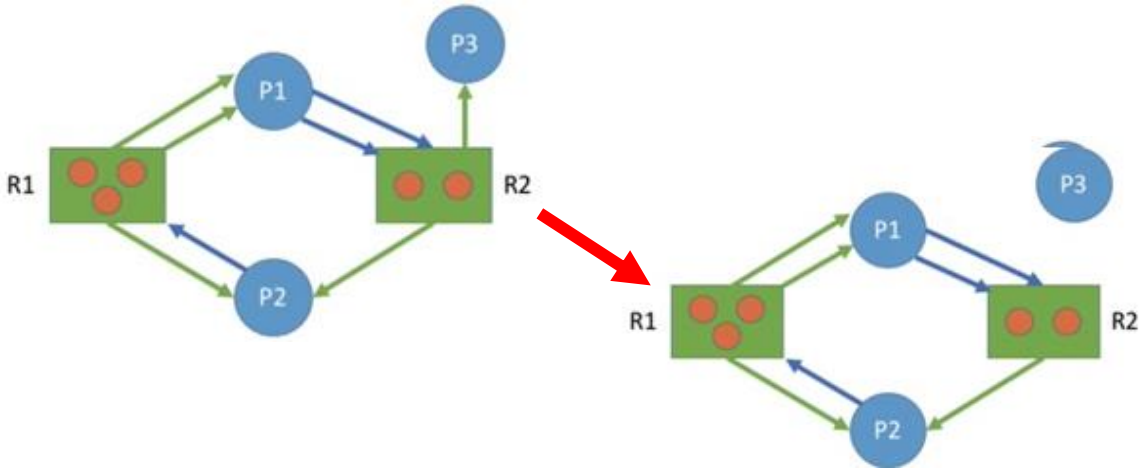
资源分配与调度——检测和解除

死锁的检测

- 如果系统中剩余的可用资源数足够满足进程的需求，那么这个进程可以顺利地执行下去。
- 如果这个进程执行结束了把资源归还系统，就可能使某些正在等待资源的进程被激活，并顺利地执行下去。如果按上述过程分析，最终能消除所有边，就称这个图是**可完全简化**的。此时一定没有发生死锁
- 如果最终不能消除所有边，那么此时就是发生了死锁。**最终还连着边的那些进程就是处于死锁状态的进程**



无死锁



有死锁

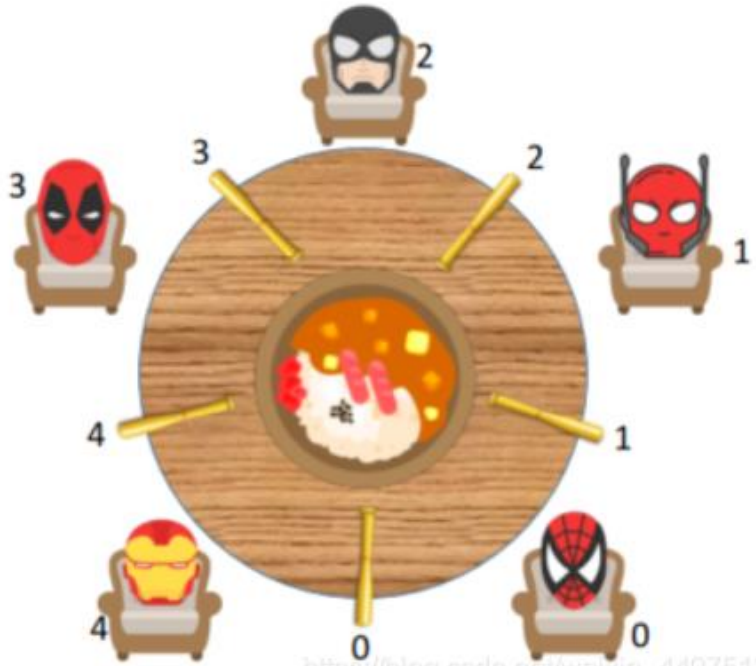
死锁的解除

- 资源剥夺法：挂起（暂时放到外存上）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。但是应防止被挂起的进程长时间得不到资源而饥饿。
- 撤销进程法（或称终止进程法）：强制撤销部分、甚至全部死锁进程，并剥夺这些进程的资源。这种方式的优点是实现简单，但所付出的代价可能会很大。因为有些进程可能已经运行了很长时间，已经接近结束了，一旦被终止可谓功亏一篑，以后还得从头再来。
- 进程回退法：让一个或多个死锁进程回退到足以避免死锁的地步。这就要求系统要记录进程的历史信息，设置还原点。

经典死锁问题——哲学家进餐问题

问题描述

- 一张圆桌上坐着5名哲学家，每两个哲学家之间的桌上摆一根筷子，桌子的中间是一碗米饭。哲学家们倾注毕生的精力用于思考和进餐，哲学家在思考时，并不影响他人。只有当哲学家饥饿时，才试图拿起左、右两根筷子（一根一根地拿起）。如果筷子已在他人手上，则需等待。饥饿的哲学家只有同时拿起两根筷子才可以开始进餐，当进餐完毕后，放下筷子继续思考。
- 尝试采用死锁预防、死锁避免、死锁检测与解除3种思路，使得餐桌上的哲学家不会被饿死。



经典死锁问题——哲学家进餐问题解答

- **死锁预防**
  - 破坏互斥条件——此路不通
  - 破坏不剥夺条件——无论改造占用者还是申请者，都容易导致“活锁”
  - 破坏请求和保持条件——采用静态资源分配法，要求哲学家必须同时拿起两根筷子
  - 破坏循环等待条件——可以采用“有序资源分配法”，对筷子进行编号，要求哲学家按照筷子编号从小到大的顺序拿起筷子（必然有一个哲学家会从右到左拿筷子）
- **死锁避免**
  - 当任意哲学家拿起任意筷子时，系统（引入一个服务员）对该动作进行审核。只有确认该动作不会导致系统死锁才允许拿起。
- **死锁检测与解除**
  - 可以设置一个“最长饥饿时间”（如吃饭时间的3倍）。如果系统中有哲学家进入吃饭状态，但超过这个时间还未吃上饭，则引入系统管理员介入，让处于饥饿状态的哲学家放下他拿起的筷子。
  - 仍然有发生“活锁”的可能，但不会一直持续。

资源分配与调度——小结

- **资源管理功能（理解）**
- **资源分配策略（理解）**
  - 先请求先服务 优先调度 针对设备特性的调度
- **死锁（掌握）**
  - 定义 举例
  - 引起死锁的原因
  - 产生死锁的必要条件
  - 死锁预防
    - 静态资源分配法
    - 有序资源分配方法
  - 死锁避免
    - 银行家算法