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5章 资源分配与调度

华中科技大学计算机学院 谢美意

目 录

CONTENT

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

资源管理概述

资源管理目标

- 保证资源的高利用率；
- 在“合理”时间内使所有顾客有获得所需资源的机会；**（饥饿）**
- 对不可共享的资源实施互斥使用；
- 防止由资源分配不当而引起的**死锁**。

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。

资源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。

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）：某些物理资源有限，采用其它物理资源改造成该类资源使用。

(2) 虚拟技术的目的

- 方便用户使用
- 资源可动态分配，提高资源利用率。

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	物理文件	逻辑文件	磁盘空间分配 文件目录查找

资源分配机构和策略

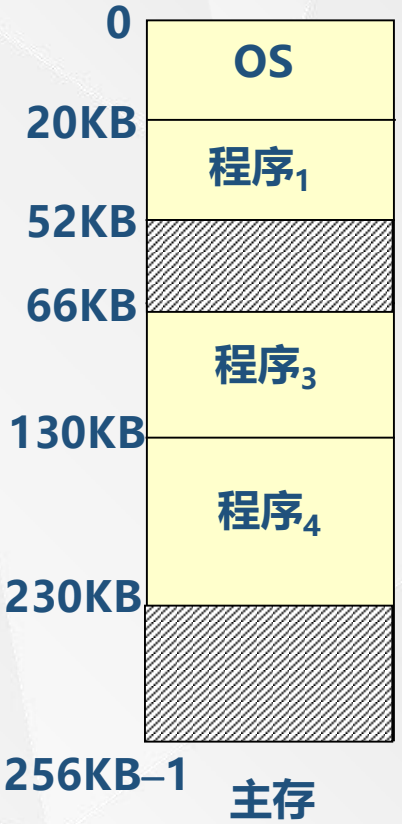
资源分配的机构

(1) 资源描述器

① 资源描述器定义

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。

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

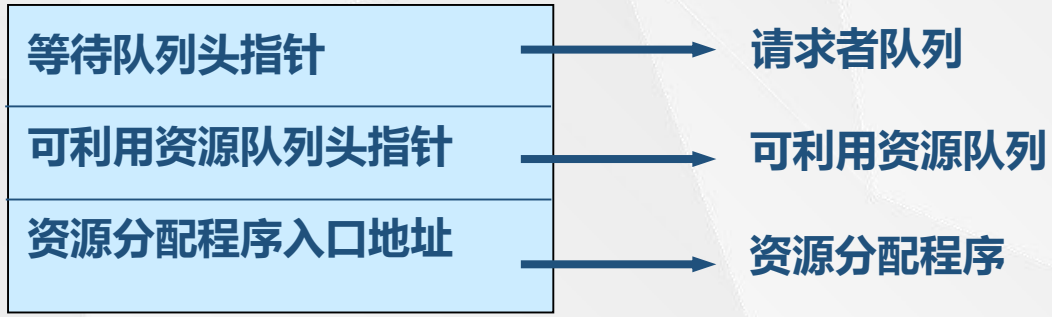
资源名、资源类型、最小分配单位的大小、地址、分配标志、描述器链接信息、存取权限、密级、存取时间。

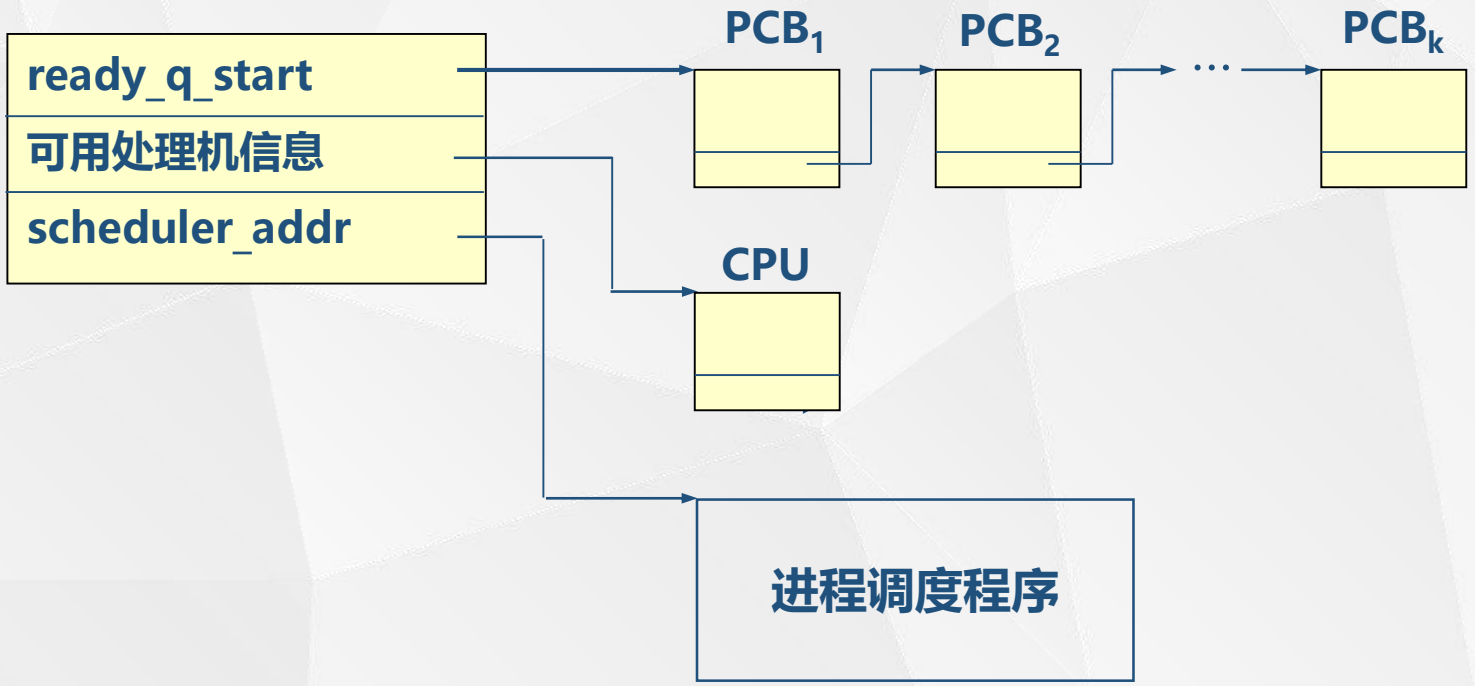
(2) 资源信息块

① 资源信息块定义

描述某类资源的请求者、可用资源和资源分配程序等必要信息的数据结构。

② 资源信息块内容



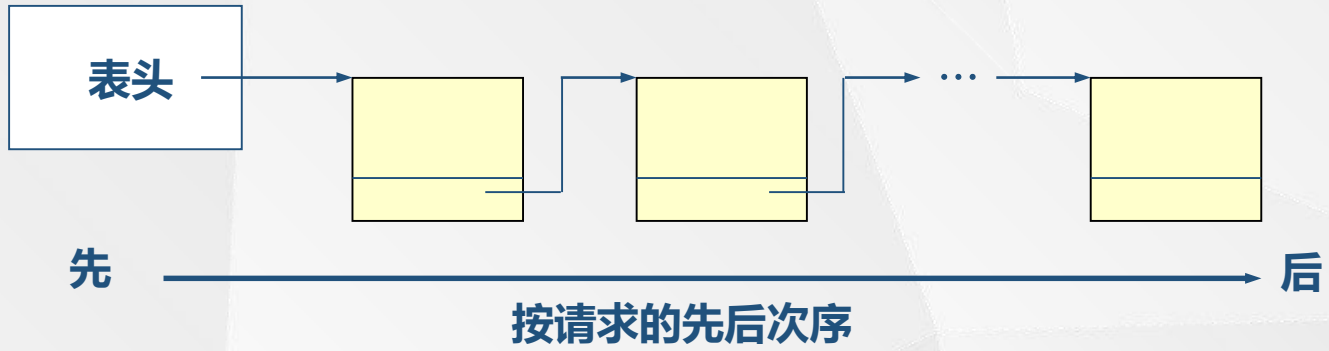

ready_q_start
可用处理机信息
scheduler_addr
PCB1
PCB2
PCBk
CPU
进程调度程序

中央处理机资源信息块

常用的资源分配策略

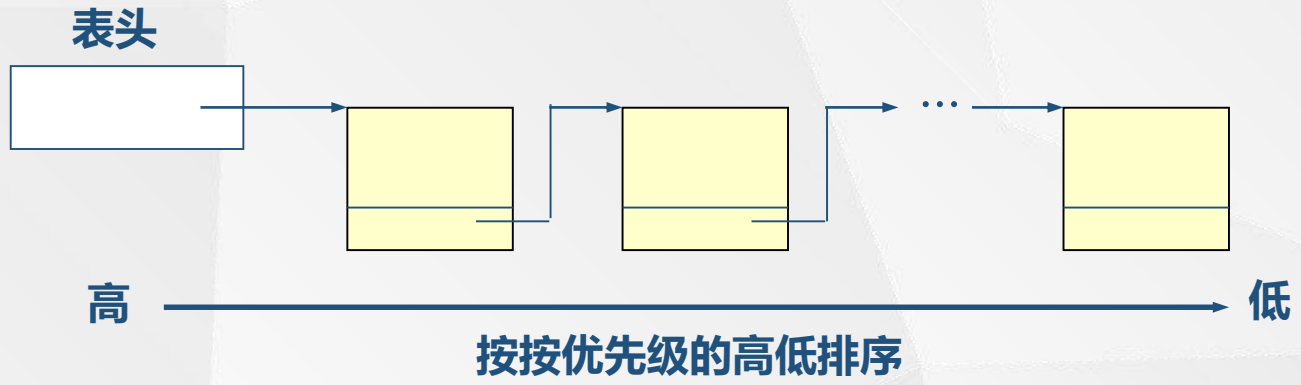
① 先请求先服务

- 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。



② 优先调度

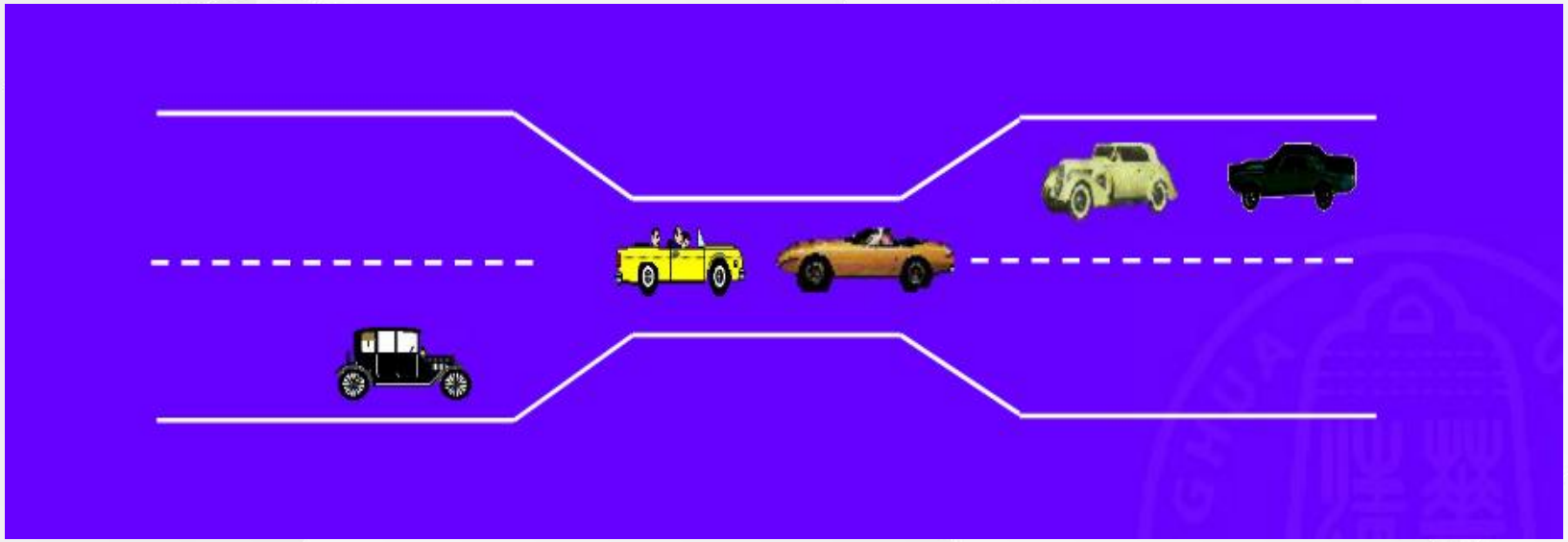
- 对每一个进程指定一个优先级；
- 每一个新产生的请求，按其优先级的高低插到相应的位置；
- 当资源可用时，取队首元素，并满足其需要。
- **排序原则**：按优先级的高低排序。



死锁

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。



死锁的例子（1）

设备共享

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是否可用，初值为1**

s_2：表示r_2是否可用，初值为1

讨论：两种资源请求序列，哪种情况可能产生死锁？

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

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
⋮	⋮
$v(s_1)$;	$v(s_2)$;
⋮	⋮
$v(s_2)$;	$v(s_1)$;
⋮	⋮

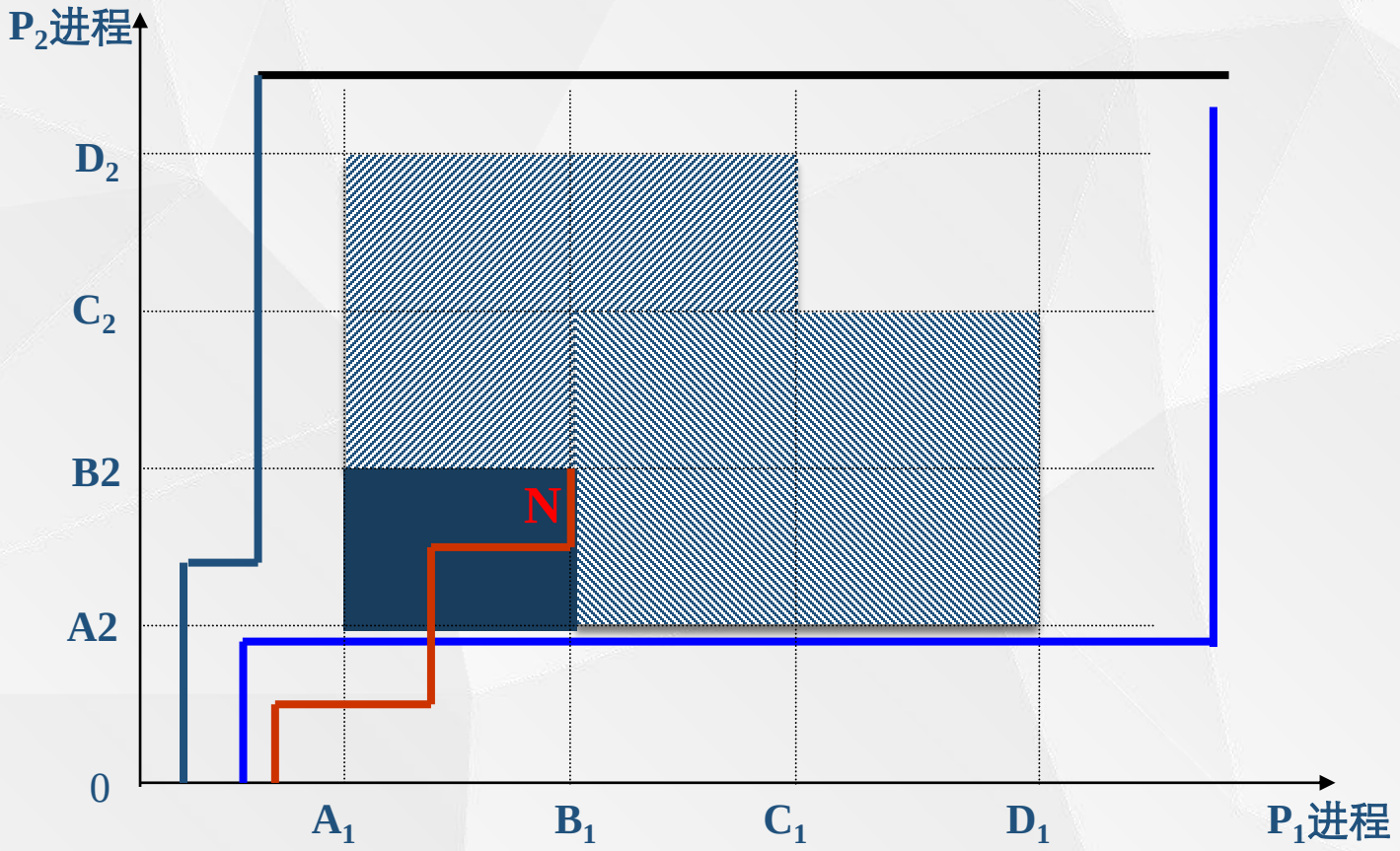
死锁的例子（2）

生产者 - 消费者问题

```
void producer()
{
    while (生产未完成) {
        ……;
        生产产品;
        P(mutex);
        P(empty);
        放入一个产品到缓冲区;
        V(P(mutex);
        V(full);
    }
}
```

- 当缓冲区满时，生产者仍可顺利执行p(mutex)操作，于是它对缓冲区有控制权，然后，当它执行p(empty)时，因没有空缓冲区被挂起。
- 能将这个生产者唤醒的是有一个消费者从缓冲区中取走一个产品，并执行v(empty)操作，但由于缓冲区已被生产者占用，消费者无法得到缓冲区的控制权，于是出现了死锁。

死锁图解



引起死锁的原因

① 系统资源不足

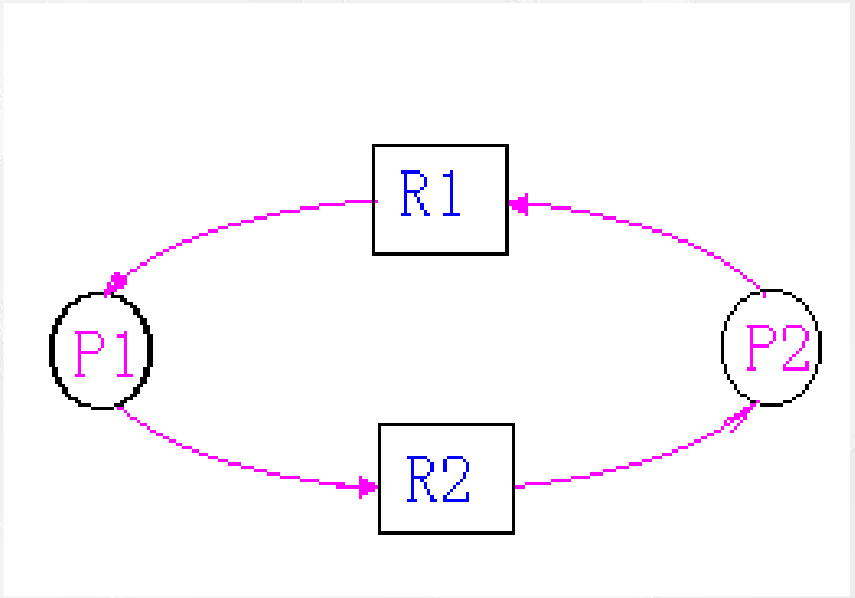
② 进程推进顺序非法

A_1: p_1 request (r_1)　B_1: p_1 request (r_2)　C_1: p_1 release (r_1)　D_1: p_1 release (r_2)

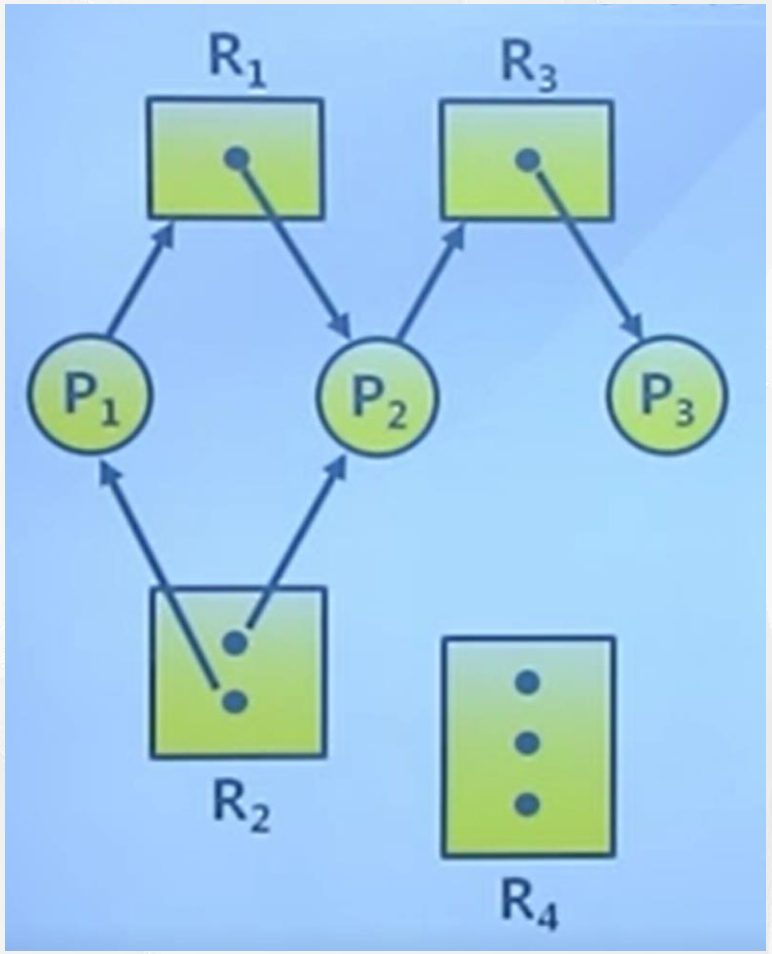
A_2: p_2 request (r_2)　B_2: p_2 request (r_1)　C_2: p_2 release (r_2)　D_2: p_2 release (r_1)

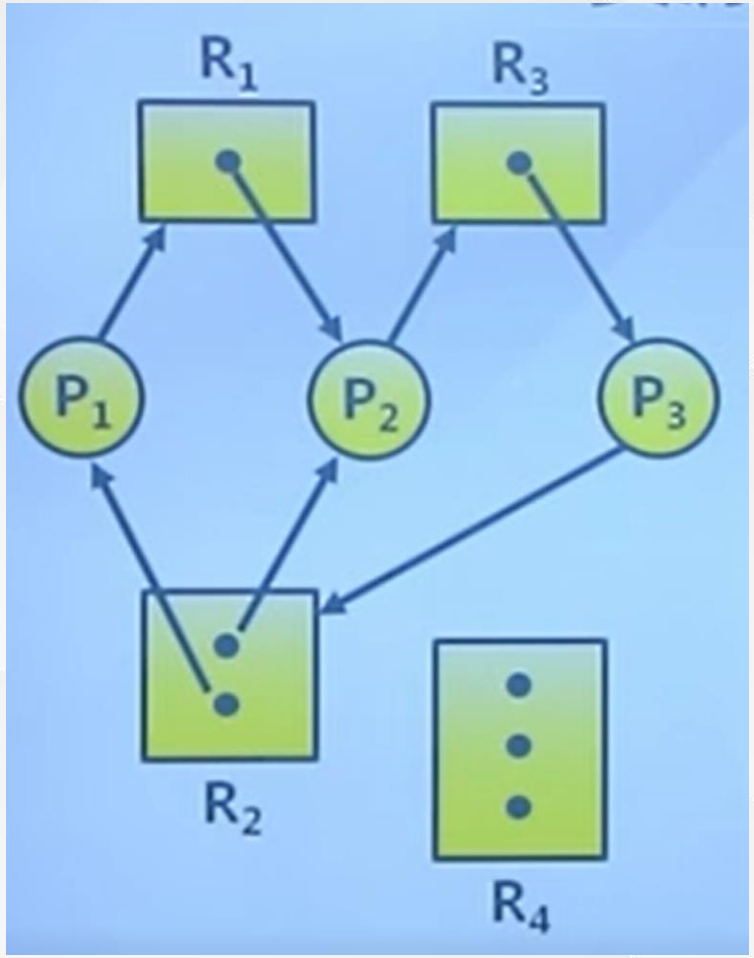
进程-资源分配图

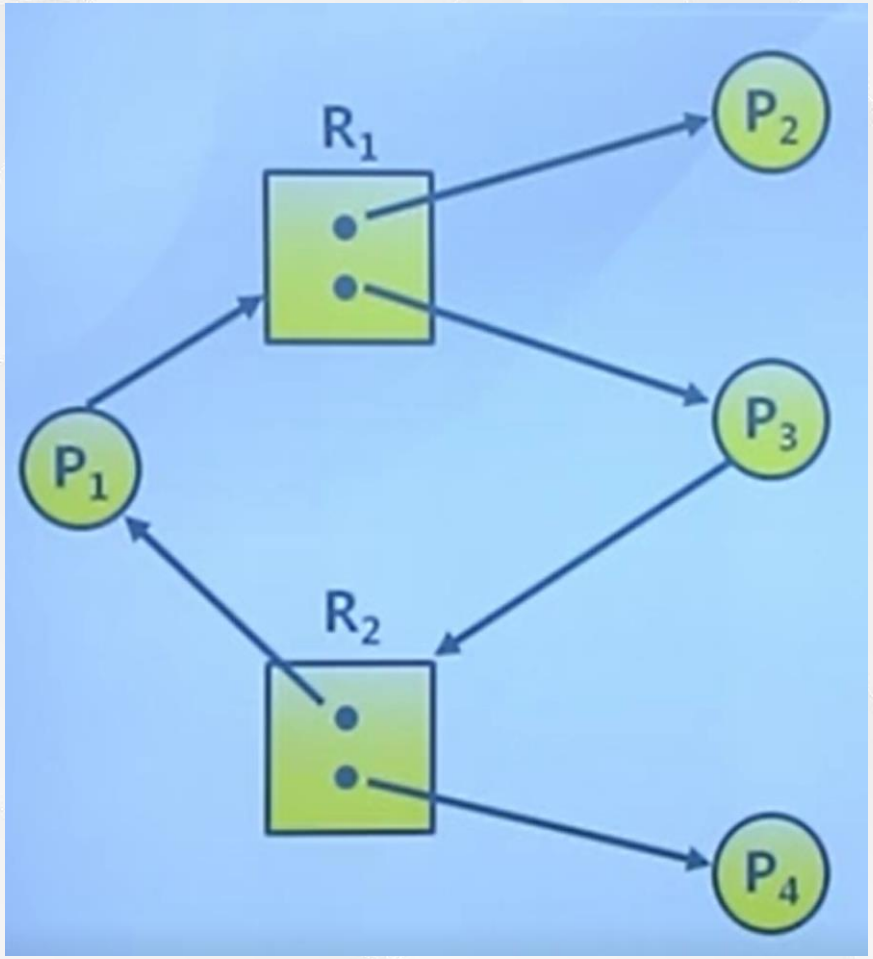
- 约定Pi→Rj为请求边，表示进程Pi申请资源类Rj中的一个资源得不到满足而处于等待Rj类资源的状态，该有向边从进程开始指到方框的边缘，表示进程Pi申请Rj类中的一个资源。
- Rj→Pi为分配边，表示Rj类中的一个资源已被进程Pi占用。



哪个图会出现死锁?







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（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。

③ 占有并等待（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（循环等待）

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。

解决死锁问题的策略

破坏产生死锁的四个必要条件之一

- 预防死锁
- 避免死锁
- 死锁的检测与解除

死锁预防——静态分配策略

在作业调度时为选中的作业分配它所需要的所有资源，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。

讨论：静态分配法破坏了四个必要条件中的哪个条件?

静态分配策略的缺点

- 一个用户（进程）在程序运行之前很难提出将要使用的全部设备；
- 设备（资源）的浪费太大，有些资源在进程运行过程中可能只有很少的时间会用到，有的甚至根本不会用到，例如，条件分支语句。

死锁预防——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。

讨论：有序资源分配法破坏了四个必要条件中的哪个条件？

有序资源分配法举例

进程PA，使用资源的顺序是R1，R2；

进程PB，使用资源的顺序是R2，R1；

采用有序资源分配法：

R1编号为1，R2编号为2

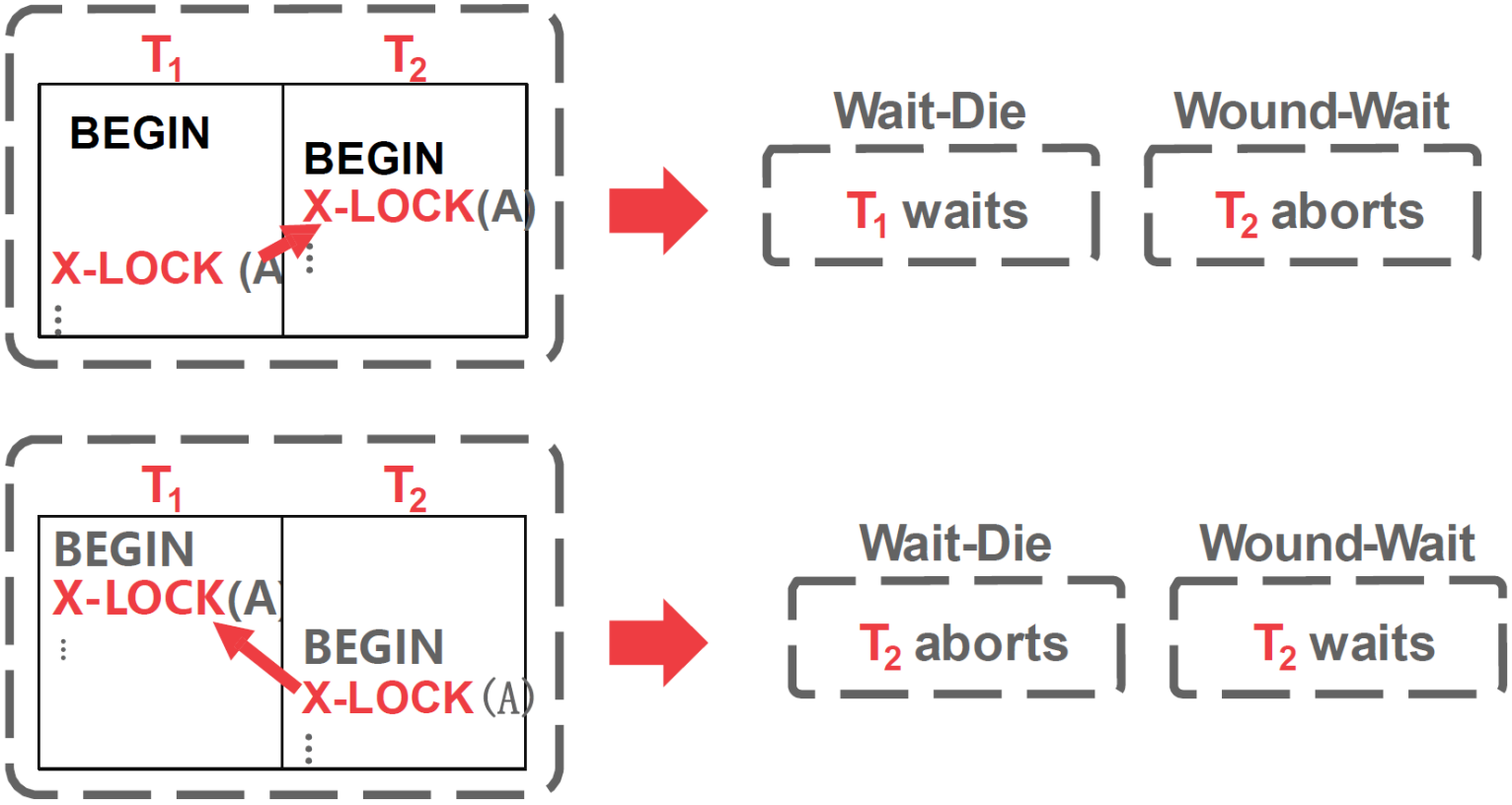
PA：申请次序应是：R1，R2

PB：申请次序应是：R1，R2

有序资源分配法的特点

- **优点：**相对于静态分配法，提高了资源使用效率
- **缺点：**进程实际使用资源的顺序不一定与资源的编号相一致。

DBMS中的死锁预防


T1
T2
BEGIN
BEGIN
X-LOCK(A)
X-LOCK (A)
Wait-Die
Wound-Wait
T1 waits
T2 aborts
T1
T2
BEGIN
X-LOCK(A)
BEGIN
X-LOCK (A)
Wait-Die
Wound-Wait
T2 aborts
T2 waits

死锁避免——银行家算法

- 银行家拥有一笔周转资金；
- 客户要求分期贷款，如果客户能够得到各期贷款，就一定能够归还贷款，否则就一定不能归还贷款；
- 银行家应谨慎的贷款，防止出现坏帐。

银行家算法思想

- 对每个资源请求进行检查，看是否会导致不安全状态。若是，则拒绝该请求；否则便满足该请求。
- 检查状态是否安全的方法是看其**是否有足够的资源满足一个距最大需求最近的客户**，如此反复下去。如果所有投资最终都被收回，则该状态是**安全状态**，最初的请求可以批准。
- 按某种顺序并发进程都能获得最大资源而顺序完成的序列为**安全序列**。

假定一个系统包括n个进程和m类资源：

(1) 资源分配矩阵

时刻 t 系统中各类资源已分配的情况，表示如下：

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

a_{ij} 表示进程p_i已占有j类资源的数目。

(2) 资源请求矩阵

时刻 t 各进程对资源的需求量，表示如下：

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

d_{ij} 表示进程p_i还需要j类资源的数目。

(3) 剩余资源向量

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

安全状态判断算法

1.寻找一个没有标记的进程p_i，对于它而言资源请求矩阵D的第i行向量小于或等于剩余资源向量R；

2.如果找到了这样的进程，则将资源分配矩阵A的第i行向量加到R中，标记该进程，并转到第1步；如果找不到这样的进程，则转到第3步；

3.如果所有进程均被标记，则系统处于**安全状态**。

银行家算法举例1

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。是否存在安全序列?

银行家算法举例2

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	5	2	1
R	3	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。是否存在安全序列?

死锁的检测和解除

对资源的分配不加任何限制，也不采取死锁避免措施。系统定时地运行一个“死锁检测”程序，判断系统内是否已出现死锁，如果检测到系统已发生了死锁，再采取措施解除它。

基于资源分配图检测系统死锁状态

① 如果进程-资源分配图中无环路，则此时系统没有发生死锁。

② 如果进程-资源分配图中有环路，且每个资源类中仅有一个资源，则系统中发生了死锁，此时，环路是系统发生死锁的充要条件，环路中的进程便为死锁进程。

③ 如果进程-资源分配图中有环路，且涉及的资源类中有多个资源，则环路的存在只是产生死锁的必要条件而不是充分条件。

死锁的解除

① 立即结束所有进程的执行，并重启操作系统。

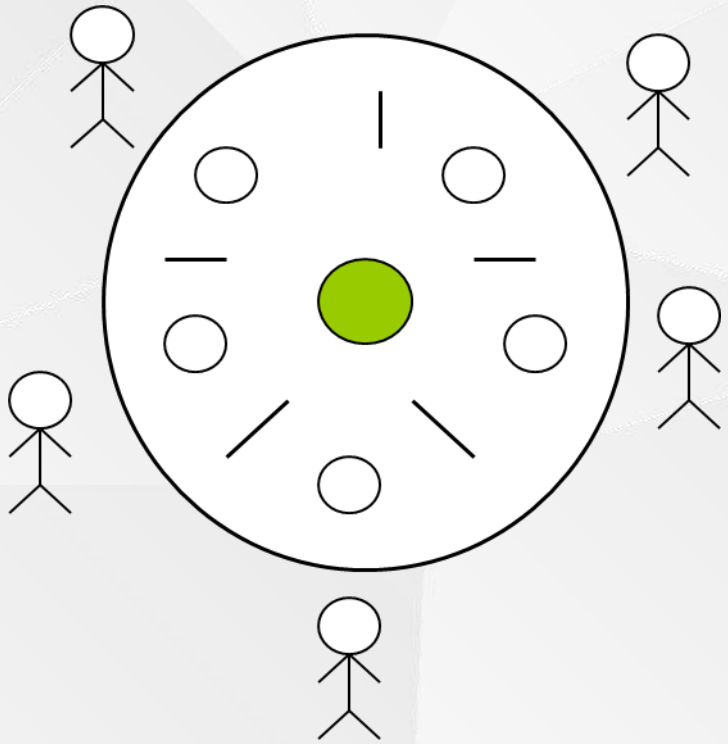
② 撤销陷于死锁的所有进程。

③ 逐个撤销陷于死锁的进程，回收其资源，直至死锁解除。

④ 剥夺陷于死锁的进程占用的资源，但并不撤销它，直至死锁解除。

⑤ 根据系统保存的checkpoint，让所有进程回退，直到足以解除死锁。

哲学家吃通心面问题



- 五个哲学家围坐在一圆桌旁，桌中央有一盘通心面，每人面前有一只空盘子，每两人之间放一把叉子。
- 每个哲学家思考、饥饿、然后吃通心面。
- 为了吃面，每个哲学家必须获得两把叉子，且每人只能直接从自己左边或右边去取叉子。

```
semaphore fork[5]={1};
cobegin
    Pi()          // i=0,1,2,3,4
    {
        while (true)
        {
            思考;
            P(fork[i]);
            P(fork[(i+1)mod 5]);
            吃通心面;
            V(fork[i]);
            V(fork[(i+1)mod 5]);
        }
    }
coend;
```

Q：此解法存在什么问题?

Q：有什么避免死锁的方法?

- 至多允许四个哲学家同时吃;
- 奇数号先取左手边的叉子，偶数号先取右手边的叉子;
- 每个哲学家取到手边的两把叉子才吃，否则一把叉子也不取。

```
semaphore fork[5]={1};
cobegin
    Pi()          // i=0,1,2,3
    {
        while (true)
        {
            思考;
            P(fork[i]);   //i=4,P(fork[0])
            P(fork[(i+1)mod 5]);  //i=4,P(fork[4])
            吃通心面;
            V(fork[i]);
            V(fork[(i+1)mod 5]);
        }
    }
coend;
```

第5章 小结

- **资源管理功能**
- **资源分配策略：**先请求先服务 /优先调度
- **死锁**
  - 死锁定义
  - 引起死锁的原因
  - 产生死锁的必要条件
  - 解决死锁问题的方法
    - 死锁预防（静态分配，有序资源分配）
    - 死锁避免（银行家算法）
    - 死锁的检测和解除