第三章 处理机调度与死锁

3.1 处理机调度层次和调度算法的目标

- **处理机调度的概念**
  - 采用一定的算法，从就绪队列中选择一个进程，让CPU来执行它。
  - 处理机分配任务由处理机调度程序完成。
- **调度的三个层次**

① 高级调度（作业调度、长程调度）

② 中级调度（内存调度、中程调度）

③ 低级调度（进程调度、短程调度）

3.1.1 处理机调度的层次

1. **高级调度**
   - **多道批处理系统特有的调度 (分时、实时没有)**
   - **作用**
     - 根据某种算法，决定把后备队列中的哪些作业调入内存，为其创建进程、分配资源，并将其插入到就绪队列。

2. 低级调度

- 三种操作系统都必须有的调度
- 作用
  - 根据某种算法，决定把CPU分配给就绪队列中的哪个进程。具体分派CPU的操作由分派程序执行。

3. 中级调度

- **引入的目的**
  - 提高内存利用率和系统吞吐量
- **作用**
  - 把暂时不运行的进程放到外存等待，留出内存给其他进程使用
  - 在外存上的进程为**“挂起状态”**
  - 当这些进程又具备运行条件、且内存有空闲时，便选择一些调入内存，进入就绪状态

3.1.2 处理机调度算法的目标

- **采用何种调度算法，主要取决于OS的类型和目标**

1. 处理机调度算法的共同目标

① **资源利用率：**尽可能使所有资源保持忙碌状态

CPU利用率 = 有效工作时间/(有效工作时间+空闲时间)

② **公平性：**保持各进程合理地获得CPU的服务

③ **平衡性：**由于进程的类型(计算型，I/O型)不同，所以使各种设备都经常处于忙绿状态

④ **策略强制执行：**对于制订的策略，即使会造成工作延迟，只要需要，也必须予以执行。

2. 批处理系统的目标

① **平均周转时间短**

周转时间：从作业提交给系统开始，到作业完成为止的这段时间间隔。

（**带权周转时间**）

② **系统吞吐量高**

吞吐量：单位时间内系统所完成的作业数

③ **处理机利用率高**

3. 分时系统的目标

① 响应时间快

响应时间：从用户(通过键盘)提交一个请求开始，直到系统给出处理结果为止的时间。

② 均衡性

均衡性：系统响应时间的快慢应与用户所请求服务的复杂性相适应

4. 实时系统的目标

① **保证截止时间**

截止时间：某任务必须开始执行的最迟时间，或必须完成的最迟时间。

② **可预测性** 通过可预测性可以进行预处理，以提高实时性

3.2 作业与作业调度

- 批处理系统中的作业
  - 作业：比程序的概念更广泛，配有一份作业说明书，以便系统进行控制。
  - 作业步：相对独立且相互关联的顺序加工步骤

作业控制块(JCB)

- 为管理和调度作业，设置JCB，是作业在系统中存在的标志。

作业运行的三个阶段和三个状态

① 收容阶段：作业提交后，保存到外存中，为其创建JCB，并加入后备队列。(后备状态)

② 运行阶段：作业被作业调度选中，为其创建进程，进入就绪队列。(运行状态)

③ 完成阶段：作业结束运行。(完成状态)

- 作业调度时要考虑的问题

① 接纳多少作业

太多——平均周转时间会显著延长

太少——不利于提高利用率和吞吐量

② 接纳哪些作业

- **先来先服务(FCFS)调度算法**
  - 最简单的算法，可用于作业调度、进程调度
  - 基本思想
    - 谁先来，先为谁服务 (类似食堂排队打饭)
  - 缺点
    - 不考虑作业(进程)的长短
    - 不考虑作业(进程)的紧迫程度

短作业(进程)优先调度算法(SJF、SPF)

- 可用于作业调度、进程调度
- **基本思想**
  - 哪个估计运行时间最短，先运行哪个
- **优点**
  - 和FCFS相比，性能有明显改善
- **缺点**
  1. 对长作业(进程)不利
  2. 没考虑作业(进程)的紧急程度
  3. 由于是估计的运行时间，所以不一定能真正做到短的优先
  4. 无法实现人-机交互(特别是对长作业、长进程)

- **优先级调度算法(PSA)**
  - **可用于作业调度、进程调度**
  - 照顾紧迫型作业，使之进入系统后能被优先处理
  - 调度算法根据优先级进行调度

◎ 高响应比优先调度算法(HRRN)

- 动态优先权

  优先权 = (等待时间+要求服务时间)/要求服务时间

  即 **响应比 = 响应时间 / 要求服务时间**

- **由上可知**

  ① 等待时间相同，要求服务时间越短，优先权越高
  ——有利于短作业

  ② 要求服务时间相同，等待时间越长，优先权越高
  ——相当于FCFS

  ③ 长作业的优先权可随等待时间增加而提高

- **优点：**短、长作业都考虑到了
- **缺点：**调度之前，须计算响应比，增加了系统开销

3.3 进程调度

- OS中必不可少的调度算法
- **进行调度的任务、机制和方式**

1. **进程调度的任务**

① 保存CPU现场

② 按某种算法选取进程

③ 把CPU分配给进程

2. 进程调度机制

为实现进程调度，在调度机制中，应具有如下三个基本部分

① 排队器

按一定策略将就绪进程进行排队，以提高调度效率

② 分派器

根据进程调度程序选中的进程，将其从就绪队列中取出，进行上下文切换，将CPU分配给该进程

③ 上下文切换

◎ 进程调度方式

① 非抢占方式

- 一旦将CPU分配给某进程，便让它一直运行，直到它完成或阻塞，才让另一个来执行
- 优点：实现简单、系统开销小，适用于大多数的批处理系统环境
- 缺点：难以满足紧急任务的要求——立即执行，因而可能造成难以预料的后果

② 抢占方式

- 允许按照某种原则暂停正在执行的进程，让另一个来执行
- 抢占的原则
  - 优先权原则
  - 短进程优先原则
  - 时间片原则

◎ 轮转调度算法

分时系统中，常用的是基于时间片的轮转(RR)调度算法

- 轮转法的基本原理

  所有就绪进程按FCFS策略排成就绪队列，系统每隔一定时间(如30ms)便产生一次中断，以激活进程调度程序进行调度

- 进程切换时机

  ① 时间片没用完，进程执行完成

  ② 时间片用完，但进程没执行完

◉ 优先级调度算法

1. 优先级调度算法的类型

① 非抢占式优先级调度算法

一旦CPU分配给就绪队列中优先级最高的进程，该进程便一直使用CPU，直到完成或因发生某事放弃CPU为止

① 抢占式优先级调度算法

某进程使用CPU期间，只要出现优先级更高的进程，调度程序就将CPU分配给该优先级最高的进程

2. 优先级的类型

确定进程的优先级、使用静态优先级还是动态优先级

① 静态优先级

- 基本思想：创建进程时分配，保持不变
- 确定优先权的依据
  - ① 进程类型：系统进程的 > 一般用户的
  - ② 进程对资源的需求：资源需求少的 > 高的
  - ③ 用户要求：根据用户的紧迫度、用户所付费用来确定
- 优点：简单易行、系统开销小
- 缺点：不精确，优先级低的可能长时间无法执行

③ 动态优先级

- 基本思想
  - 随执行或等待时间的增加而改变，如
  - 优先级低的等待一定时间后，优先级会提高
  - 规定当前执行的进程优先级随时间下降
- 优点
  - 在抢占方式下，可防止长进程长期霸占CPU

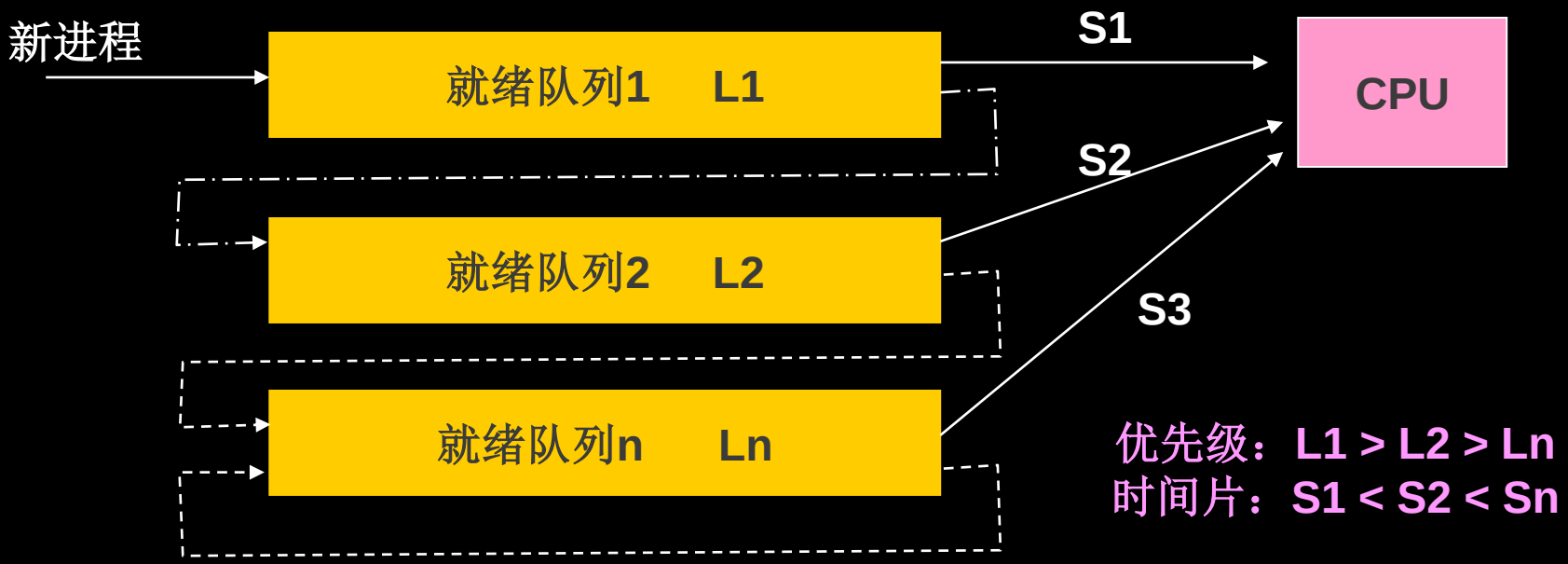
多级反馈队列调度算法

- 基本思想

① 设多个就绪队列(按优先级划分)，优先级越高，获得的时间片越小

② 新进程进入系统后，放入第一个队列，按FCFS原则等待。第i个时间片执行完后，便放入i+1队列尾

③ 仅当队列L1—Li 空时，才从Li+1队列中调度进程——可以抢占



- **多级反馈队列调度算法的性能**
  ① 性能较好
  ② 能较好满足各类用户的需求
  ③ 短进程可在一两个时间片内完成
  ④ 长进程也不必担心长期得不到处理

3.4 实时调度

- 实时系统与其他系统的最大区别
  - 处理和控制的正确性不仅取决于计算的结果，还取决于计算和处理结果产生的时间
- 实时系统具有的特点
  - 进程往往带有一定的紧迫度
  - 实时系统中，任务往往联系着一个截止时间
  - 所以，要引入一种新的调度——“实时调度”

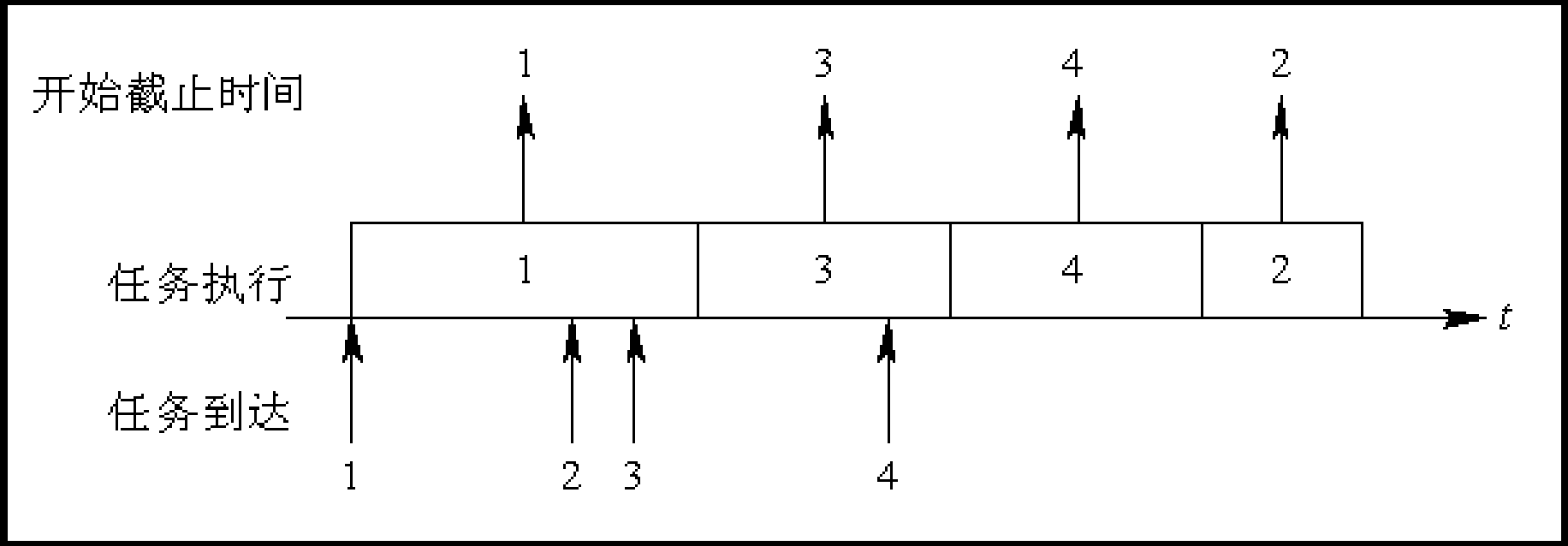
- 实现实时调度的基本条件
  1. **提供必要的信息**
     - **就绪时间**：任务何时进入就绪状态的？
     - **截止时间**：开始截止时间、完成截止时间
     - **处理时间**：任务从开始执行到完成所需时间
     - **资源要求**：需要哪些资源
     - **优先级**：描述进程的紧迫度
  2. **系统处理能力强**
     - 如果CPU处理能力弱，可能导致某些任务不能及时被处理
  3. **采用抢占式调度机制**
     - 硬实时任务的实时系统中，广泛采用抢占方式
  4. **具有快速的切换机制**
     - 保证任务能快速切换

- **实时调度算法的分类**
  1. 非抢占式调度算法
     - 应用于小型系统、要求不严的系统
  2. 抢占式调度算法
     - 应用于要求严格的系统

1. 非抢占式调度算法
   - ① 非抢占式轮转调度算法
     - 计算机控制多个相同的对象，为每个创建一个实时任务，排成轮转队列。用于工业生产的群控系统。
   - ② 非抢占式优先调度算法
     - 用于有一定要求的实时控制系统
2. 抢占式调度算法
   - ① 基于时钟中断的抢占式优先级调度算法
     - 时钟中断到来后才进行抢占，有较好的响应效果。用于大多数的实时控制
   - ② 立即抢占的优先级调度算法
     - 只要当前任务未处于临界区可立即抢占，可获得非常快的响应。用于发生外部中断的紧急任务

常用的几种实时调度算法

1. 最早截止时间优先算法(EDF)
   - 根据截止时间确定优先级
     - 越早，优先级越高
   - 就绪队列按优先级排序
   - 可用于抢占、非抢占调度



EDF用于非抢占调度方式

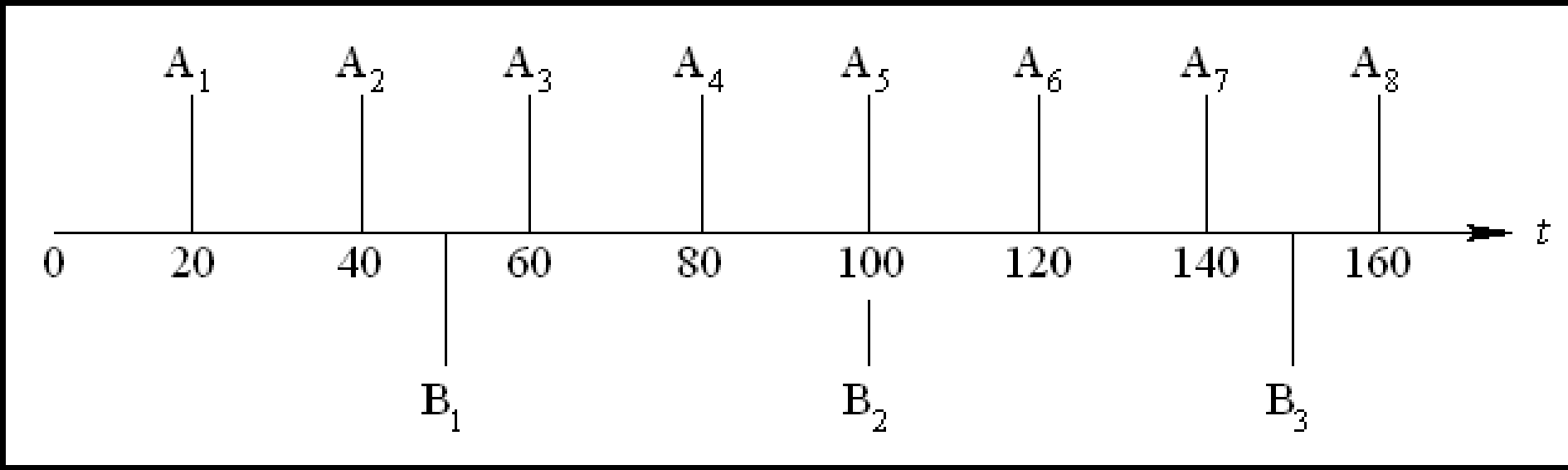
开始截止时间：3早于2，4早于2

2. 最低松弛度优先(LLF)算法

- 主要用于抢占调度方式
- 根据任务紧急(或松弛)的程度，松弛度越低(紧急度越高)，优先级越高

松弛度 = 完成截止时间 - 剩余运行时间 - 当前时间

例：两个周期性实时任务A、B，
A：每 20 ms执行一次，执行时间为 10 ms
B：每 50 ms执行一次，执行时间为 25 ms



A和B任务每次必须完成的时间

① 在t= 0 ms时

A_1松弛度 = 20 – 10 – 0 = 10

B_1松弛度= 50 – 25 – 0 =25

故应先调度A_1执行。

② 在 t=10 ms时，

A_2的松弛度= 必须完成时间-剩余运行时间-当前时间

= 40 - 10 - 10 = 20

B_1的松弛度= 必须完成时间-剩余运行时间-当前时间

= 50 - 25 -10 = 15

故应先调度B_1执行。

3.5 死锁概述

3.5.1 资源问题

1. 可重用性资源、消耗性资源

1) 可重用性资源：用户可重复使用多次，其性质：

① 每个该资源中的单元只能分给一个进程，不允许多进程共享

② 进程使用资源时，须按照顺序：请求资源、使用资源、释放资源

③ 每一类可重用性资源中的单元数目相对固定，进程运行期间既不能创建也不能删除

- 对资源的请求和释放通常利用系统调用实现

2) 可消耗性资源：临时性资源，进程运行期间，由进程动态创建和消耗，其性质：

① 每个该资源的单元数目在运行期间可以不断变化

② 进程运行期间，可以不断创造这类资源

③ 进程运行过程中，可以请求若干这类资源单元，用于进程自己消耗，用后不再返回到该资源类中

- 通常由生产者进程创建，由消费者进程消耗
- 典型实例：进程间通信的消息

2. 可抢占性资源、不可抢占性资源

1) 可抢占性资源

某进程获得这类资源后，该资源可被其他进程或系统抢占，

这类资源不会引起死锁（典型实例：CPU、内存）

2) 不可抢占性资源

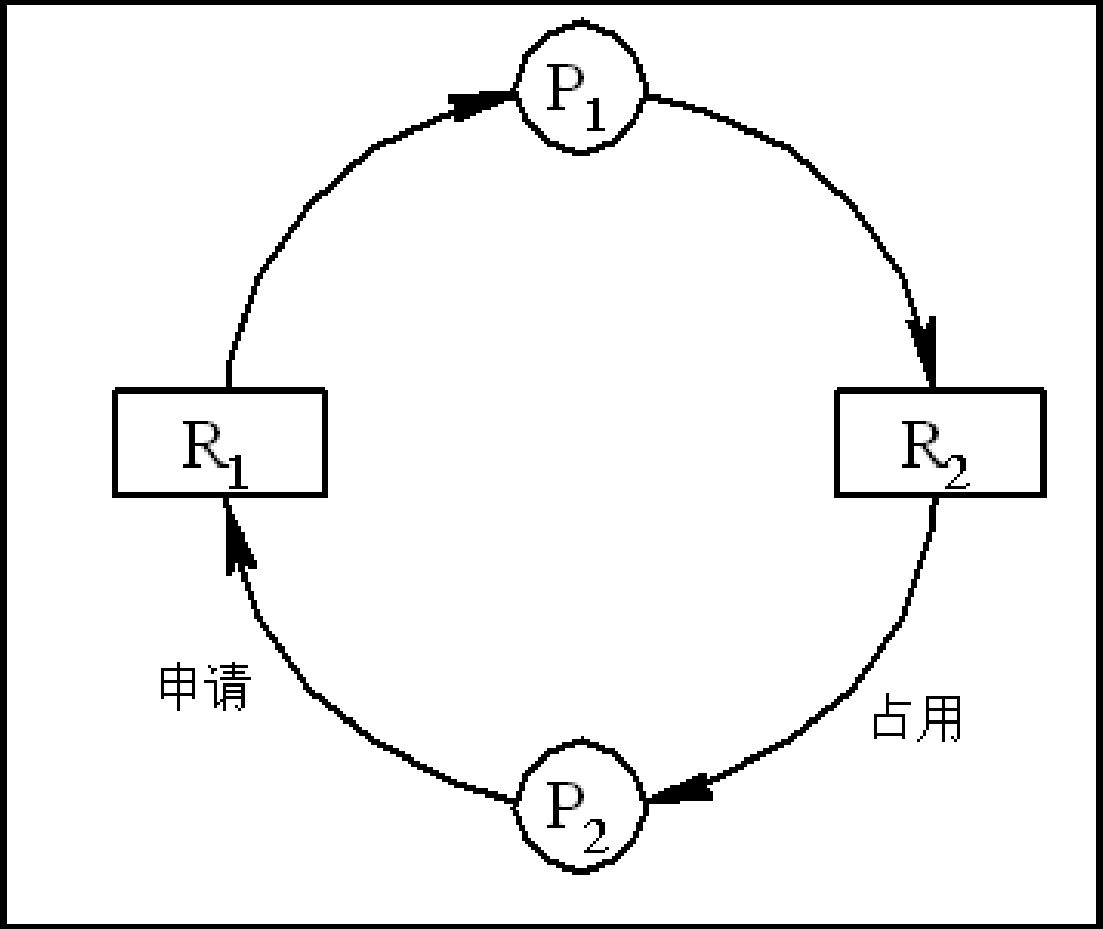
一旦系统把该类资源分配给某进程，就不能将它强行收回，

只能在进程用完后自行释放（典型实例：打印机）

3.5.2 计算机系统中的死锁

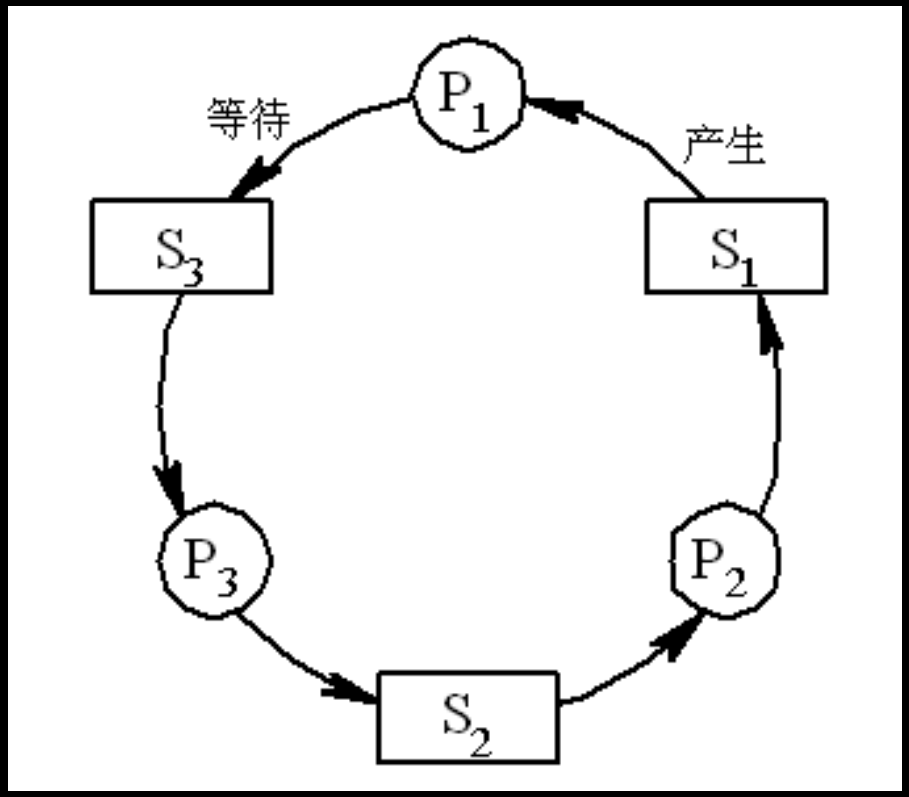
- 死锁的起因
  - 对不可抢占资源的争夺
  - 对可消耗资源的争夺

1. 竞争不可抢占性资源引起死锁



I/O设备共享时的死锁情况

2. 竞争可消耗资源引起死锁



进程之间通信时的死锁

3. 进程推进顺序不当引起死锁

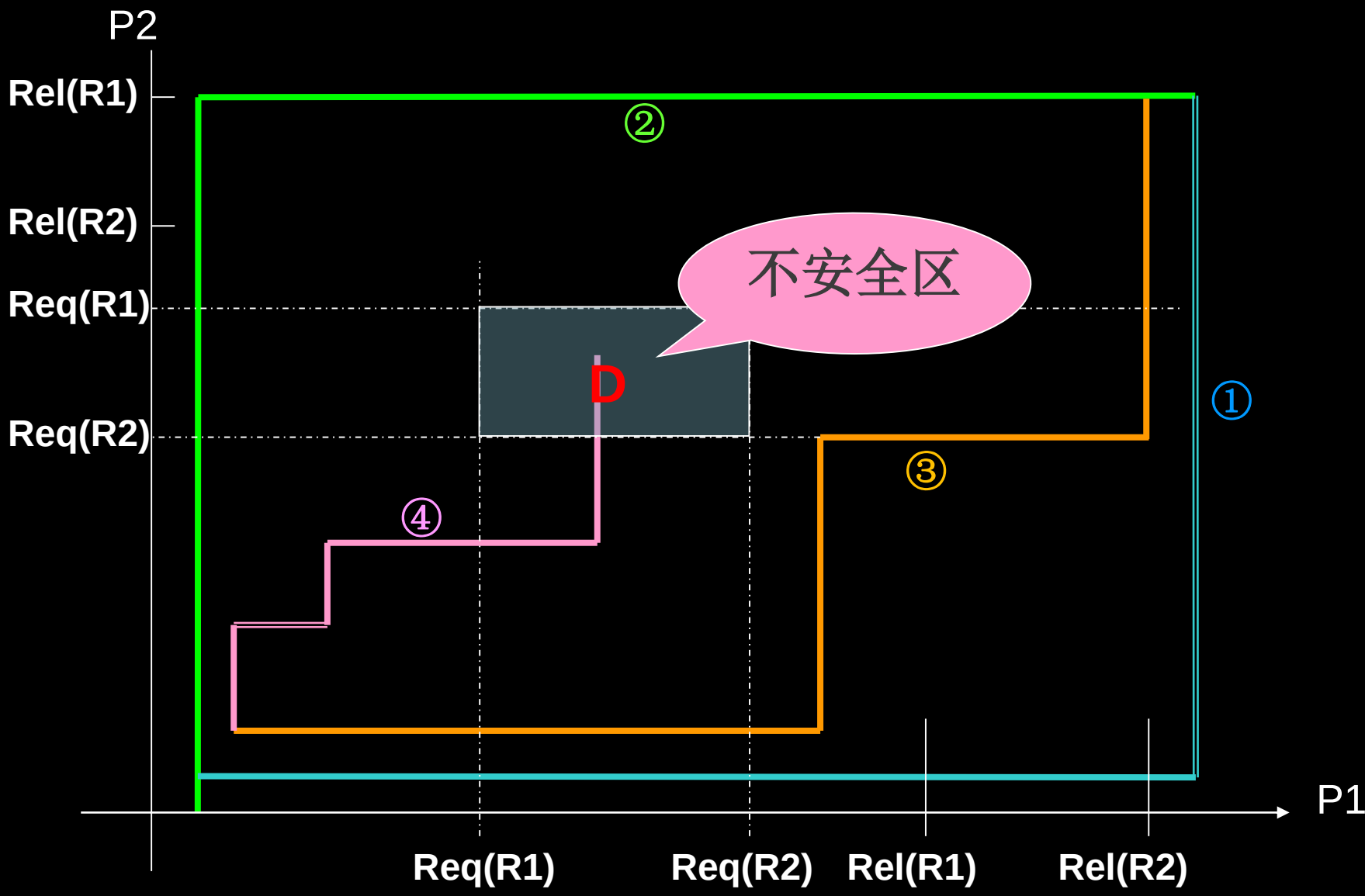


图 3-14 进程推进顺序对死锁的影响

3.5.3 死锁的定义、必要条件、处理方法

- 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。

- 产生死锁的必要条件
  1. 互斥条件
     - 进程在使用某资源时，不允许其他进程使用
  2. 请求和保持条件
     - 进程已保持至少一个资源，请求另一个资源时无法获得，便阻塞，但又不放弃已占用的资源
  3. 不可抢占条件
     - 资源使用完前，不允许被抢占，直至其用完
  4. 循环等待条件
     - 发生死锁时，一定存在 “ 进程-资源”循环链
- **注：只有四个条件同时满足时，才会死锁**

处理死锁的基本方法

① **预防死锁**

- 设置限制条件，破坏四个条件中的一个或几个
- 简单易行，使用广泛

② **避免死锁**

- 动态分配资源时，防止系统进入不安全状态

③ **检测死锁**

- 不用任何限制条件，也不检查系统是否进入安全区。允许发生死锁。可通过机制及时检测出死锁

④ **解除死锁**

- 常用方法：撤销一些进程，以便回收它们的资源，再分给阻塞的进程

3.6 预防死锁

- 基本思想
  - 使四个条件中的2、3、4不成立
  - 条件1(互斥条件)是设备固有的，所以不应限制，反而应加强

3.6.1 破坏“请求和保持”条件

- 基本思想
  - 规定所有进程在开始之前，都必须一次性申请其在整个运行过程中所需的全部资源
  - 只要有一种资源不够，便全不分配
- 优点
  - 简单易行，安全
- 缺点
  - 资源浪费
  - 进程延迟执行 (仅当全部资源都获得后，才能运行)

- 改进思想
  - 允许进程只获得运行初期所需的资源后，便开始运行
  - 运行过程中再逐步释放已获得、且已用完的全部资源
  - 然后再请求新的资源
- 优点
  - 提高设备利用率
  - 进程可更快执行

② 破坏“不可抢占”条件

- 基本思想
  - 进程可以逐个申请资源，一旦申请的资源无法满足，立即释放已经保持的所有资源
- 缺点
  - 复杂
  - 代价大 (资源被迫释放，可能导致以前的工作无效)
  - 反复申请资源，可能使进程执行被无限推迟

③ 破坏“循环等待”条件

- 基本思想
  - 按类型给资源赋序号
  - 进程申请资源必须按序号递增次序提出
  - 如：输入机=1,打印机=2,磁带机=3,磁盘=4
- 优点
  - 利用率、吞吐量明显改善
- 缺点
  - 资源序号相对稳定，不利新设备加入
  - 进程使用资源次序和序号次序不同，浪费
  - 对用户编程不利

3.7 避免死锁

3.7.1 系统安全状态

1. 安全状态

◎ 定义

- 系统能按某顺序(P1, P2, …，Pn)，为每个进程Pi分配所需资源，直至满足每个进程对资源的最大需求，使每个进程都可顺利完成
- 称 (P1, P2, …, Pn) 序列为**安全序列**
- 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——可能导致死锁

◎ 避免死锁实质

- 进行资源分配时，如何使系统不进入不安全状态

2. 安全状态举例

- 设有进程P_1、P_2和P_3，及12台磁带机
- P_1、P_2、P_3分别需要10、4、9台磁带机
- T_0时刻，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
- 请问T_0时刻是否是安全的？
- 解答：实质是问，T_0时刻是否存在一个安全序列

	进程	最大需求	已分配	可用
2	P_1	10	5	12
1	P_2	4	2	
3	P_3	9	2	

安全序列(P_2,P_1,P_3)

3. 由安全状态向不安全状态的转换

- 如果不按照安全序列分配资源，系统可能会由安全状态进入不安全状态。
- 比如：P_3又请求1台磁带机以后：

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

不安全了！

- 所以，尽管系统尚有可用的磁带机，也不能分配给P_3

3.7.2 利用银行家算法避免死锁

- 提出者：Dijkstra
- 最著名的避免死锁的算法
- 基本思想
  - 当用户申请一组资源时，系统要判断如果把这些资源分配出去，系统是否还处于安全状态。
  - 若是，就可以分配；
  - 否则，该申请暂不予满足

1. 银行家算法中的数据结构

① 可利用资源向量Available[m]

- 表示m种资源中每种可供使用的数量，初始值是系统中每类资源的全部数量

Available [j] =K 表示目前资源 j 有 K 个可供使用

② 最大需求矩阵Max[n,m]

- 表示n个进程中的每个进程对m种资源的最大需求量

Max [i,j] =K 表示进程 i 对资源 j 的最大需求量为 K

③ 已分配矩阵Allocation[n,m]

- 每种资源已分配给每个进程的数量

Allocation［i,j］=K, 进程i当前已获得资源 *j* 的数目为*K*

④ 需求矩阵Need[n,m]

- 每个进程还需要资源的数量

Need［i,j］=*K*, 表示进程i还需要*K*个资源 j ，才能执行完

- **数据结构的小结**

① Available［j］：目前资源 j可供使用的数量

② Max［i,j］：进程i需要资源 j 的最大数量

③ Allocation［i,j］：进程i当前已获得资源 j 的数量

④ Need［i,j］：进程i还需要多少资源 j，才能执行完

•所以，Need［i,j］= Max［i,j］- Allocation［i,j］

2. 银行家算法

- 设向量$Request_i[m]$
- 如果$Request_i$［j］=*K*，表示进程P_i需要*K*个资源 j
- P_i申请资源时，OS如下进行检查：

① 如果**$Request_i$［j］≤ Need［i,j］**，执行下一步；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］≤Available［j］**，执行下一步；否则， 表示尚无足够资源，P_i须等待。

③ OS**假设**把资源分配给进程P_i，并作如下修改：

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 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**真正**把资源分配给进程P_i；否则，本次假设的分配作废，**恢复**原来的资源分配状态，让进程P_i等待。

3. 安全性算法

- 作用
  - 查看是否存在一个安全序列
- 算法

1. 设置两个数组向量

① **工作向量 Work[m]**

- OS可提供给进程继续运行所需的各种资源数量
- 安全性算法初始时，Work = Available;

② **Finish[n]**

- 系统是否有足够的资源分配给进程，使之运行完成。
- 初始时 Finish［i］= false;
- 当有足够资源分配给进程 i 时， 便令 Finish［i］= true

2. 从所有的进程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. 执行以下两步修改

- Finish［i］= true;
- Work［j］= Work［j］+ Allocation［i,j］;
- **因为进程P_i获得所需的资源后，便可畅通无阻的执行完，并释放出已分配给它的资源，从而使系统里的可用资源增多。**

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

4. 银行家算法举例

- 五个进程{P_0, P_1, P_2, P_3, P_4}和三类资源{A, B, C}，各种资源的数量分别为10、5、7，在T_0时刻的资源分配情况如图所示

进程 \ 资源情况	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
										(2	3	0)
P_1	3	2	2	2	0	0	1	2	2			
				(3	0	2)	(0	2	0)			
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			

1. T_0时刻的安全性：

进程 \ 资源情况	Work A	Work B	Work C	Need A	Need B	Need C	Allocation A	Allocation B	Allocation C	Work+Allocation A	Work+Allocation B	Work+Allocation C	Finish
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

2. P1请求资源

- P1发出请求向量Request1(1，0，2)，系统按银行家算法进行检查

① Request1(1, 0, 2) ≤ Need1(1, 2, 2)

② Request1(1, 0, 2) ≤ Available (3, 3, 2)

③ 系统先假定可为P1分配资源，并修改Available, Allocation1和Need1向量，由此形成的资源变化情况如图中的圆括号所示

④ 再利用安全性算法检查此时系统是否安全

进程 \ 资源情况	Work A B C	Need A B C	Allocation A B C	Work+Allocation A B C	Finish
P_1	2 3 0	0 2 0	3 0 2	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_0	7 4 5	7 4 3	0 1 0	7 5 5	true
P_2	7 5 5	6 0 0	3 0 2	10 5 7	true

P_1申请资源时的安全性检查

存在安全序列：(P1，P3，P4，P0，P2)

3. P4请求资源

- P4发出请求向量Request4(3，3，0)，系统按银行家算法进行检查

① Request4(3, 3, 0) ≤ Need4(4, 3, 1);

② Request4(3, 3, 0) > Available(2, 3, 0)，让P4等待

4. P0请求资源

- P0发出请求向量Requst0(0，2，0)，系统按银行家算法进行检查

① Request0(0, 2, 0) ≤ Need0(7, 4, 3);

② Request0(0, 2, 0) ≤ Available(2, 3, 0);

③ 系统先假定可为P0分配资源，并修改有关数据，如下图所示

进程 \ 资源情况	Allocation A B C	Need A B C	Available A B C
P_0	0 3 0	7 2 3	2 1 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

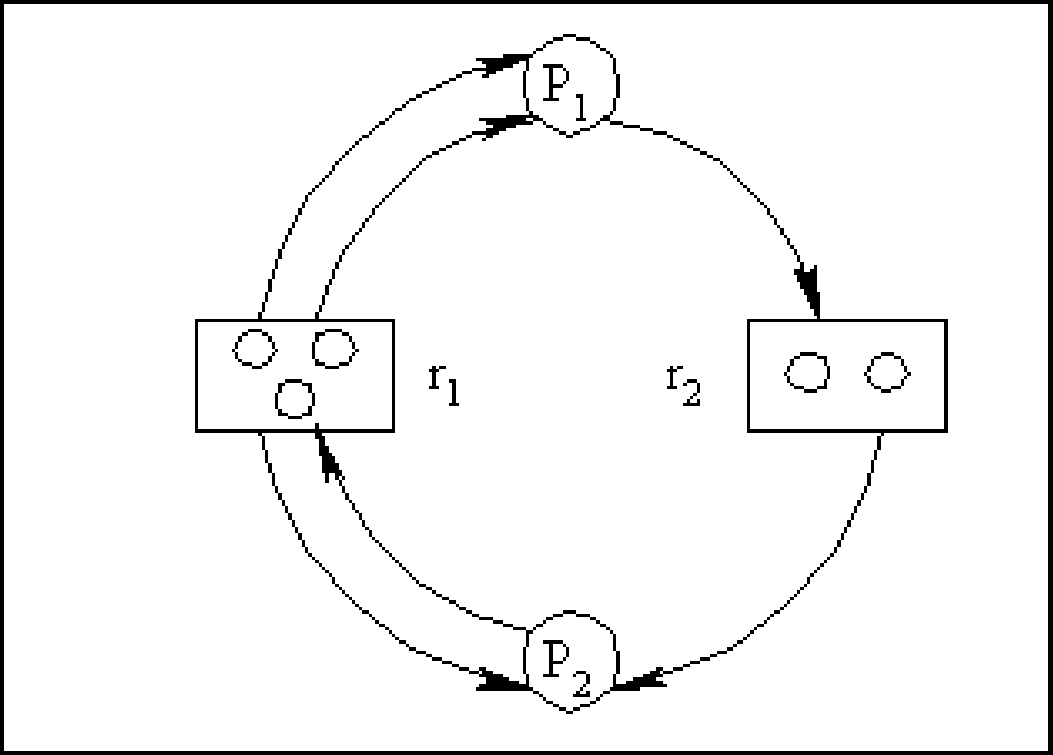
为P_0分配资源后的有关资源数据

- 已不能满足任何进程的需要，所以无法进入安全状态，此时不能分配资源

3.8 死锁的检测与解除

3.8.1 死锁的检测

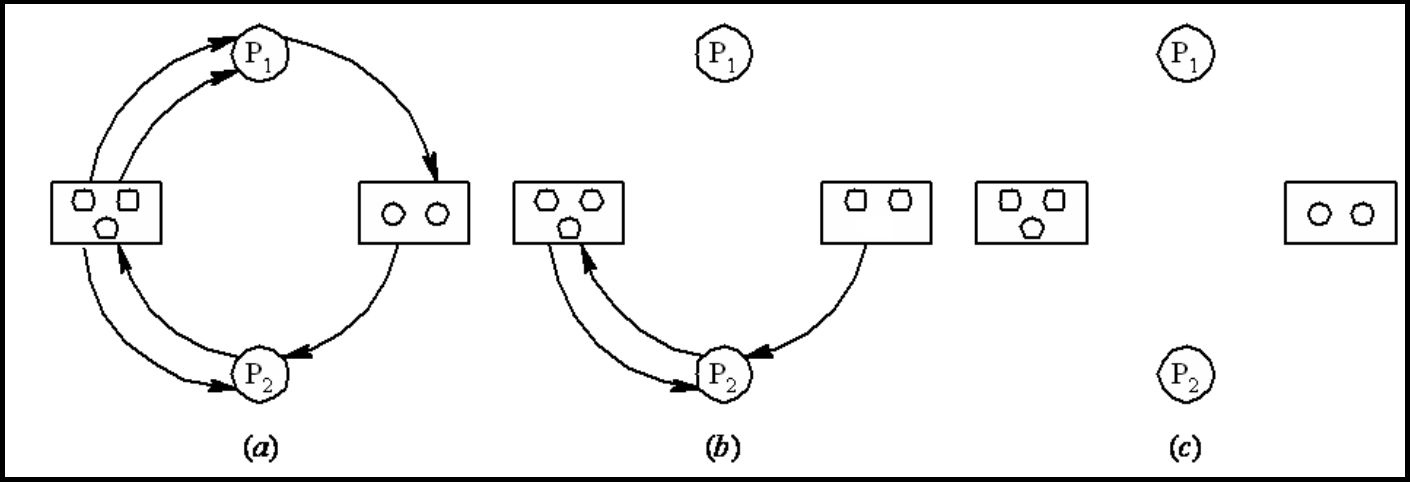
- 资源分配图



每类资源有多个时的情况

死锁定理

- 基本思想
  - 通过简化资源分配图, 检测死锁
- 基本方法
  - 不断寻找既不阻塞又不独立的进程结点Pi
  - 顺利时，Pi可获得所需资源，并执行完
  - 相当于，消去连接Pi的各条边，使之成为孤立结点
- 死锁定理
  - 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



3.8.2 死锁的解除

- 两种方法

① 抢占资源

- 从进程抢占足够资源分给死锁进程

② 终止进程（有两种方法）

a) 终止全部死锁进程

b) 按某顺序逐个终止死锁进程，直到有足够资源为止

小结

- 三种级别的调度(高、中、低)
- 调度和死锁的基本概念
- **常用的调度算法**
- **死锁概念、产生的必要条件(四个条件)**
- 死锁的预防和避免方法
- **银行家算法**
- 死锁的检测及解除

练习题

1. 避免死锁的一个著名的算法是(B)

A. 先入先出法； B. 银行家算法；

C. 优先级算法； D. 资源按序分配法

2. 在一般操作系统中必不可少的调度是(D)

A. 高级调度； B. 中级调度；

C. 作业调度； D. 进程调度

3. 资源的有序分配算法在解决死锁问题中是用于（ A ）

A. 预防死锁　　B. 避免死锁

C. 检测死锁　　D. 解除死锁

4. 一进程在获得资源后，只能在使用完资源时由自己释放，这属于死锁必要条件的（ C ）

A. 互斥条件　　B. 请求和释放条件

C. 不可抢占条件　D. 循环等待条件

5. 在下列进程调度算法中，哪一个算法会对优先级进行调整（ C ）

A. 先来先服务　　B. 短进程优先

C. 高响应比优先　D. 时间片轮转

6. 下面（ A ）算法不是进程调度算法

A. LRU　B. FCFS　C. SJF　D. RR

7. 既考虑作业等待时间，又考虑作业执行时间的调度算法是（ A ）

A. 高响应比优先　B. 短作业优先

C. 优先级调度　　D. 先来先服务

8. 作业调度算法从(D)队列中选取适当的作业投入运行

A. 执行　B. 就绪　C. 完成　D. 后备

9. (A) 是指从作业提交给系统到作业完成的时间间隔

A. 周转时间　B. 响应时间

C. 等待时间　D. 运行时间

10. 下述作业调度算法中，(B) 调度算法与作业的估计运行时间有关

A. 先来先服务　B. 短作业优先

C. 均衡　　D. 时间片轮转

11. 采用资源剥夺法可解除死锁，还可以采用（ B ）方法解除死锁

A. 执行并行操作　　B. 撤消进程

C. 拒绝分配新资源　D. 修改信号量

12. 产生死锁的四个必要条件是：互斥、（ B ）、循环等待和不可抢占

A. 请求与阻塞　B. 请求与保持

C. 请求与释放　D. 释放与阻塞

13. 发生死锁的必要条件有四个，要防止死锁的发生，可以破坏这四个必要条件，但破坏(　A　)条件是不太实际的

A. 互斥　B. 不可抢占　C. 请求与保持　D.循环等待

14. 银行家算法是一种(　B　)算法

A. 解除死锁　B. 避免死锁

C. 预防死锁　D. 检测死锁

15. 当进程数大于资源数时，进程竞争资源(　B　)会产生死锁

A. 一定　B. 不一定

C. 不可能　D. 以上答案都不对

16. (B)优先级是在创建进程时确定的，确定之后在整个进程运行期间不再改变

A. 先来先服务　B. 静态　C. 动态　D. 短作业

17. OS中，出现死锁指的是(C)

A. 计算机发生故障

B.资源数少于进程数

C. 若干进程因竞争资源而无限等待其他进程释放已占用的资源

D. 进程同时申请的资源数超过资源总数

18. 哪种说法对抢占式调度来说是正确的(C)

A. 系统采用轮转调度，则系统采用的是非抢占式调度

B. 若当前进程等待某一事件时引起调度，则该系统采用的是抢占式调度

C. 实时系统通常采用抢占式调度

D. 采用抢占式调度的系统中，进程的周转周期较非抢占式的系统是可预测的

19. 3个进程各需要4个同类资源，问该资源最少要有几个才不会产生死锁(B)

A. 9　B. 10　C. 11　D.12

进程	已分配	可用
P1	3	1
P2	3	
P3	3	

考虑资源分配时最坏的情况：所有进程在执行过程中，都只申请资源，不释放。

20. 现有8台打印机，N个进程争用，每个可能用3台，N最大为多少时没有死锁的危险
（ C ）

A. 1　B. 2　C. 3　D. 4

进程	已分配	可用
P1	2	1
P2	2	
P3	2	

N*(3-1) + 1 = 8
所以　N = 7/2 = 3.5，取整得 3

判断题

- 进程实体由PCB和数据集，及对数据集进行操作的程序组成 （ √ ）
- 并发是并行的不同表述，原理相同（ X ）
- 所谓多道程序设计，指每一时刻可以有若干个进程在执行 （ X ）
- 银行家算法用来检测死锁 （ X ）
- 只要破坏产生死锁的四个必要条件中的其中一个就可以预防死锁的发生 （ √ ）

填空题

- 系统中有n个进程，就绪队列中最多可有(n - 1)个进程
- 不让死锁发生的策略可分为静态、动态两种，避免死锁属于（动态策略 ）
- 系统中有m个进程，死锁涉及了k个进程，问k应满足的取值范围(2 ≤ k ≤ m)