操作系统
Operating System

汤臣薇、冯文韬

tangchenwei@scu.edu.cn, Wtfeng2021@scu.edu.cn

四川大学计算机学院（软件学院）

数据智能与计算艺术实验室

第三章

处理机调度与死锁

处理机调度与死锁

- 处理机调度的层次和调度算法的目标
- 作业与作业调度 
- 进程调度
- 实时调度
- 死锁概述
- 预防死锁 
- 避免死锁
- 死锁的检测与解除

回顾——处理机调度的层次

◆ 高级调度/长程调度/作业调度

- 在一个(批)作业运行完毕退出系统，而需重新调入一个(批)作业进入内存时发生，周期较长，大约几分钟一次

◆低级调度/短程调度/进程调度/微观调度

- 运行频率最高，时间尺度毫秒级，分时系统（10～100 ms 进行一次）
- 频繁使用，要求在实现时做到高效

◆中级调度/中程调度/内存调度

- 运行频率基本上介于上述两种调度之间

回顾——处理机调度算法的目标

◆ 处理机调度算法的共同目标

- 资源利用率
- 公平性
- 平衡性
- 策略强制执行

◆ 批处理系统的目标

- 作业周转时间和平均周转时间短
- 系统吞吐量高
- 处理机利用率高

◆ 分时系统的目标

- 响应时间快（重要准则）
- 均衡性

◆ 实时系统的目标

- 截止时间的保证
- 可预测性
- 实时性

回顾——批处理系统中的作业

◆ 作业（Job）——比程序更为广泛的概念

- ➢ 作业控制块 JCB(Job Control Block)
- ➢ 一个系统能够接纳作业的个数称为系统的多道程序度
- ➢ 作业状态
  - 后备状态：磁盘输入井，等待调入内存运行
  - 运行状态：被作业调度程序选中后，分配必要的资源，建立一组相应的进程后，调入内存，进程各状态（运行态、就绪态、阻塞态、外存就绪态、外存阻塞态等）都对应作业运行状态。
  - 完成状态：正常运行结束或因发生错误而终止时

回顾——作业调度算法

- **先来先服务调度算法（最简单）FCFS**
  - ✓利于 CPU 繁忙型的作业，适合长作业
- **短作业优先调度算法（较常用）SJF/SPF**
  - ✓适合短作业

$$R_P = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}} = \frac{\text{响应时间}}{\text{要求服务时间}}$$

- **基于作业优先级调度算法（较常用）PSA**
  - ✓基于作业的紧迫程度，由外部赋予作业响应的优先级
- **响应比高者优先调度算法（比较好）HRRN/HRRF**
  - ✓既照顾了短作业，又考虑了作业到达的先后次序

回顾——进程调度的任务、机制和方式

◆ 进程调度方式

➢ 非抢占方式 (Nonpreemptive Mode)

- 一旦把处理机分配给某进程后，让它一直运行下去，直至该进程完成，或发生某事件而被阻塞时，才再把处理机分配给其他进程
- **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

➢ 抢占方式 (Preemptive Mode)

- 允许调度程序根据某种原则去暂停某个正在执行的进程，将已分配给该进程的处理机重新分配给另一进程
- **优点**：防止长进程长时间占用处理机，为大多数进程提供更公平的服务，能满足对响应时间有着较严格要求的实时任务的需求
- **缺点**：比非抢占方式调度所需付出的开销较大

回顾——进程调度算法

- 先来先服务调度算法（最简单）FCFS
- 短进程优先调度算法（较常用）SPF
- 轮转调度算法 Time Round-Robin, TRR/RR
- 优先级调度算法 Priority Scheduling Algorithm, PSA
- 多队列调度算法 Multi-level Queue
- 多级反馈队列调度算法 Multi-level Feed Queue
- 基于公平原则的调度算法

回顾——实时调度算法

- **最早截止时间优先 (Earliest Deadline First, EDF)**
  - 可用于抢占式调度，也可用于非抢占式调度方式
- **最低松弛度优先(Least Laxity First, LLF)算法**
  - 根据任务紧急(或松弛)的程度，来确定任务的优先级
  - 主要用于可抢占的调度方式中
  - 松弛程度 = 必须完成时间 - 其本身的运行时间 - 当前时间
- **优先级倒置(priority inversion problem)现象的形成**
  - 高优先级进程/线程被低优先级进程/线程延迟或阻塞
  - 简单方案：低优先级进程进入临界区后不允许抢占处理机
  - 实用方法：动态优先级继承

处理机调度与死锁

- 处理机调度的层次和调度算法的目标
- 作业与作业调度 
- 进程调度
- 实时调度
- 死锁概述
- 预防死锁 
- 避免死锁
- 死锁的检测与解除

处理机调度与死锁

- 资源问题
- 计算机系统中的死锁
- 死锁的定义、必要条件和处理方法

死锁概述

预防死锁 

避免死锁

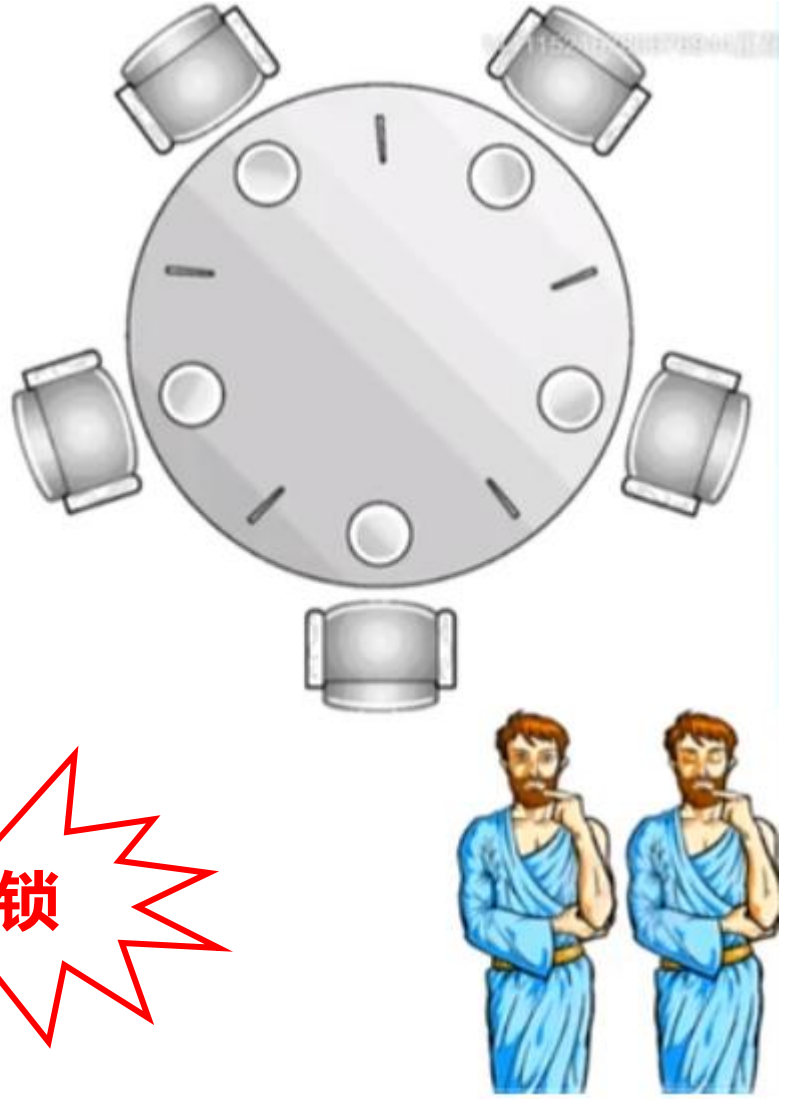
死锁的检测与解除

3.5 死锁概述

哲学家进程

```
#define  N  5
void philosopher(int i)
{
   While(TRUE)
  {
      think（）；
      take_forks(i);
      take_forks((i + 1) % N);
      eat();
      put_forks(i);
      put_forks((i + 1) % N);
  }
}
```

死锁



3.5.1 资源问题

- **引起死锁的主要是需要采用互斥访问方法的、不可被抢占的资源，及临界资源（打印机、数据文件）**
  - **可重用性资源：用户重复使用多次的资源**
  - **可消耗性资源/临时性资源：进程动态创建和消耗**
  - **可抢占/剥夺性资源**
  - **不可抢占/剥夺性资源**

3.5.1 资源问题

- **可重用性资源：用户重复使用多次的资源**
  - 每个单元只能分配给一个进程使用
  - 按照一定顺序：请求资源 → 使用资源 → 释放资源
  - 每类资源的单元数目相对固定，进程运行时不能创建或删除
  - 对资源请求或释放利用系统调用实现
    - 设备：request/release
    - 文件：open/close
    - 互斥访问的资源：wait/signal

3.5.1 资源问题

- **可消耗性资源/临时性资源：进程动态创建和消耗**
  - 每一类的单元数目在进程运行期间可变化
  - 进程运行过程可不断创造可消耗资源单元，放入缓冲区
  - 进程运行过程可请求若干个可消耗资源单元，并不再返回
  - 生产者进程创建，消费者进程消费
  - 典型：用于进程通信的消息

3.5.1 资源问题

◆ 可抢占/剥夺性资源

- ➢ 进程在获得这类资源后，该资源可再被其他进程或系统剥夺
- ➢ CPU和主存均属于可剥夺性资源

◆ 不可抢占/剥夺性资源

- ➢ 当系统把这类资源分配给某进程后，再不能强行收回，只能在进程用完后自行释放
- ➢ 磁带机、打印机属于不可剥夺性资源

3.5.2 计算机系统中的死锁

◆ 可重用性资源

◆ 可消耗性资源/临时性资源

◆可抢占/剥夺性资源

◆不可抢占/剥夺性资源

◆导致死锁的原因

- 竞争不可抢占性资源引起死锁
- 竞争可消耗性资源引起死锁
- 进程推进顺序不当引起死锁

3.5.2 计算机系统中的死锁

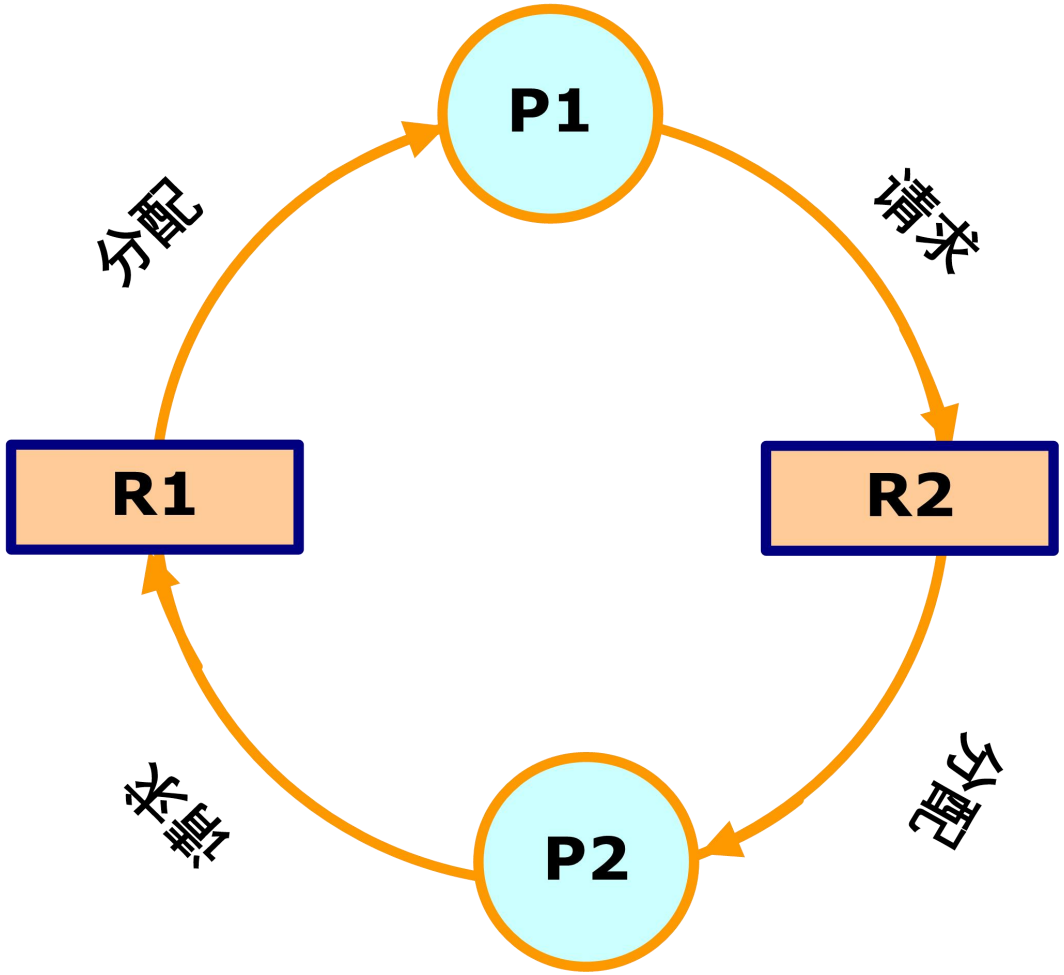
◆ 竞争不可抢占性资源引起死锁

- 系统所配置的**不可抢占性**资源由于数量不能满足诸进程运行的需要，会使进程在运行过程中因争夺资源而**死锁**
- 系统中只有一台打印机R1和一台磁带机R2，可供进程P1和P2共享。假定P1已占用了打印机R1，P2已占用了磁带机 R2。此时，若 P2 继续要求打印机，P2将阻塞；P1 若又要求磁带机，P1也将阻塞。于是，在P1与P2之间便形成了**僵局**，两个进程都在**等待对方释放出自己所需的资源**。但它们又都因**不能继续获得**自己所需的资源而**不能继续推进**，从而也不能释放出自己已占有的资源，以致进入**死锁状态**。

3.5.2 计算机系统中的死锁

◆ 竞争不可抢占性资源引起死锁

- 系统中只有一台打印机R1和一台磁带机R2，可供进程P1和P2共享。假定P1已占用了打印机R1，P2已占用了磁带机 R2。此时，若 P2 继续要求打印机，P2将阻塞；P1 若又要求磁带机，P1也将阻塞。于是，在P1与P2之间便形成了**僵局**，两个进程都在**等待对方释放出自己所需的资源**。但它们又都因**不能继续获得**自己所需的资源而**不能继续推进**，从而也不能释放出自己已占有的资源，以致进入**死锁状态**。



3.5.2 计算机系统中的死锁

◆ 竞争可消耗资源引起死锁

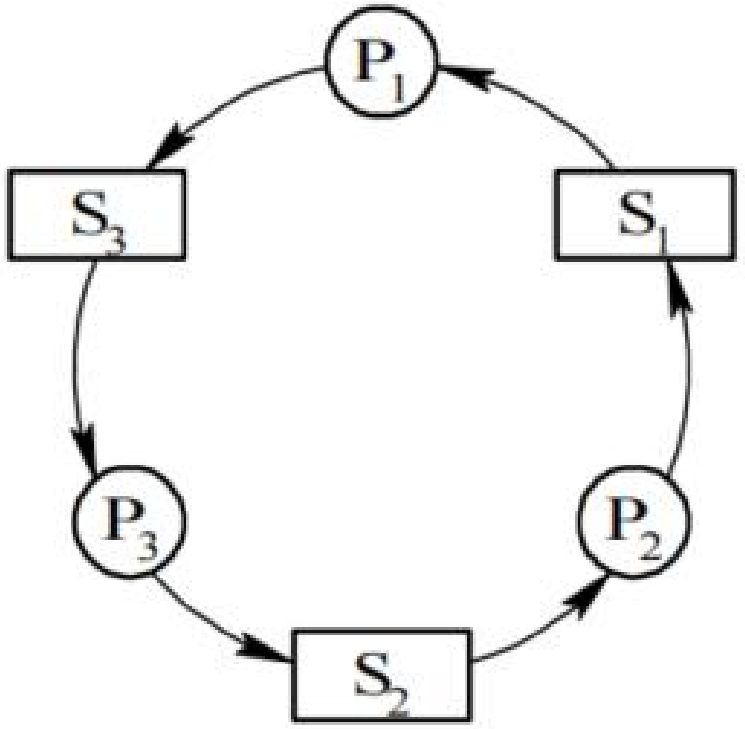
- 由一个进程产生被另一进程使用一短暂时间后便无用的可消耗资源也可能引起死锁。
- S1、S2和S3是临时性资源。进程P1产生S1，从P3接收 S3；进程P3产生消息S3，从进程P2接收S2；进程P2产生S2，从P1接收S1。
- 如果消息通信按下述两种顺序进行：

P1：…Release(S1)；Request(S3)；…
P2：…Release(S2)；Request(S1)；…
P3：…Release(S3)；Request(S2)；…

P1：…Request(S3)；Release(S1)；…
P2：…Request(S1)；Release(S2)；…
P3：…Request(S2)；Release(S3)；…

3.5.2 计算机系统中的死锁

- **竞争可消耗资源引起**
  - 由一个进程产生被…的可消耗资源也可能引起死锁。
  - S1、S2和S3是临时…收 S3；进程P3产生消息S3，从进程P2接收…
  - 如果消息通信按下…



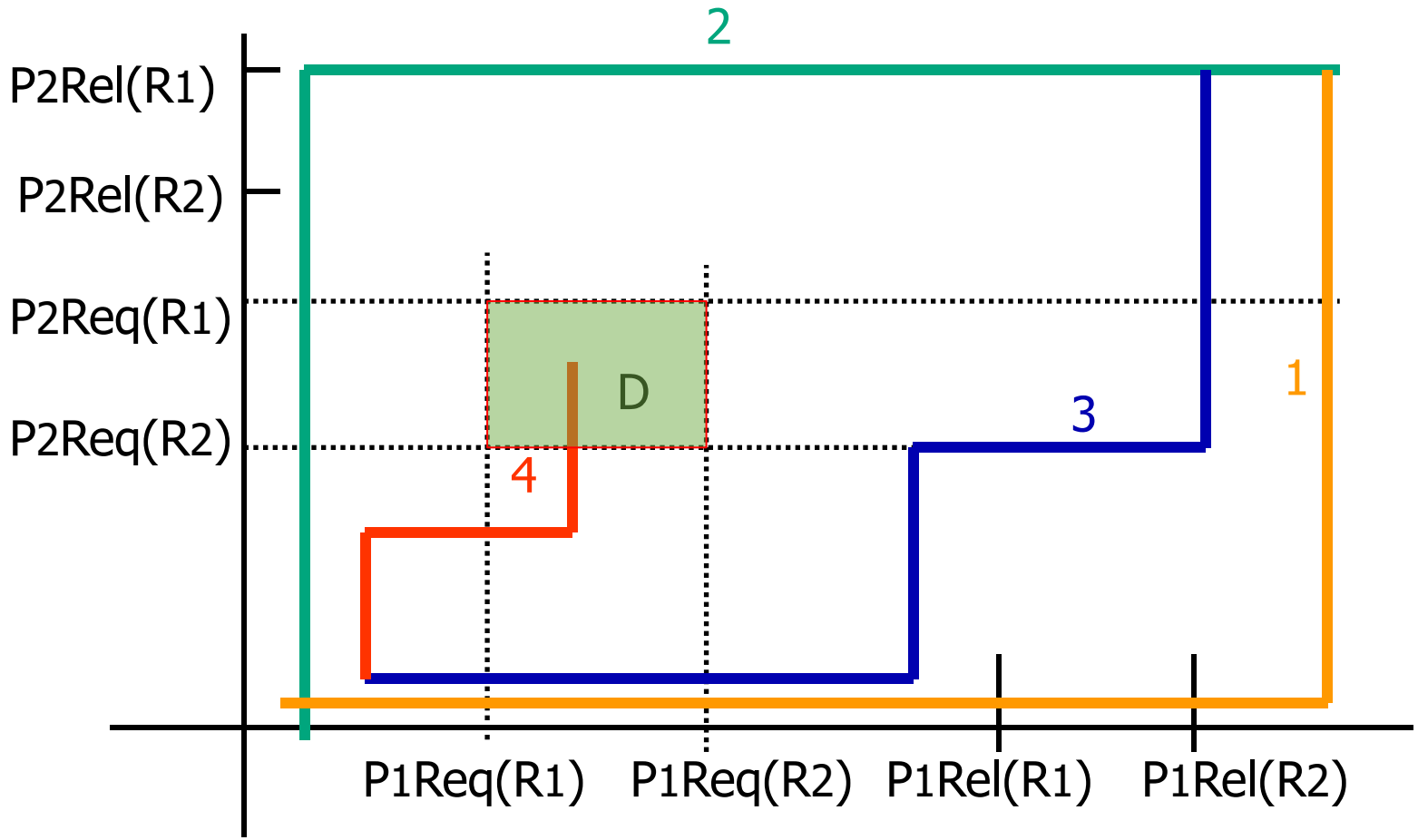
P1：…Release(S1)；Reqaest(S3)；…
P2：…Release(S2)；Request(S1)；…
P3：…Release(S3)；Request(S2)；…

P1：…Request(S3)；Release(S1)；…
P2：…Request(S1)；Release(S2)；…
P3：…Request(S2)；Release(S3)；…

3.5.2 计算机系统中的死锁

◆ 进程推进顺序不当引起死锁

- ➢ 进程在运行中具有异步性特征



3.5.2 计算机系统中的死锁

◆ 进程推进顺序不当引起死锁

- **进程推进顺序合法**：在进程P1和P2并发执行时，按照上图曲线①②③所示顺序推进时，两进程会顺利完成

  P1：Request(R1)；Request(R2)；

  P1：Releast(R1)；Release(R2)；

  P2：Request(R2)；Request(R1)；

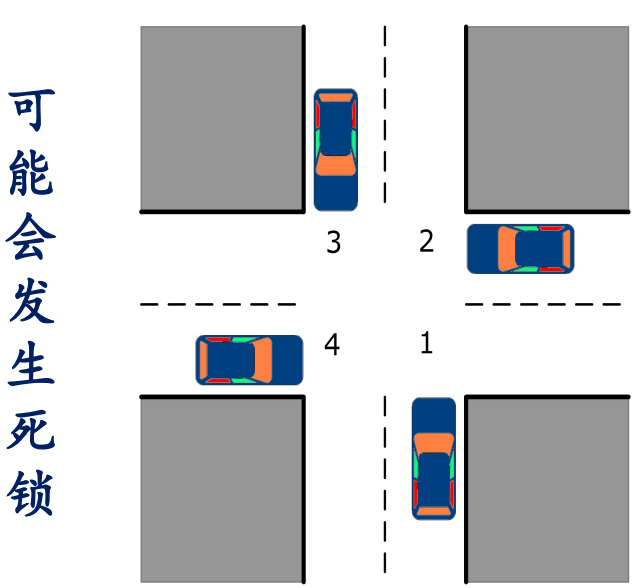
  P2：Release(R2)；Release(R1)；

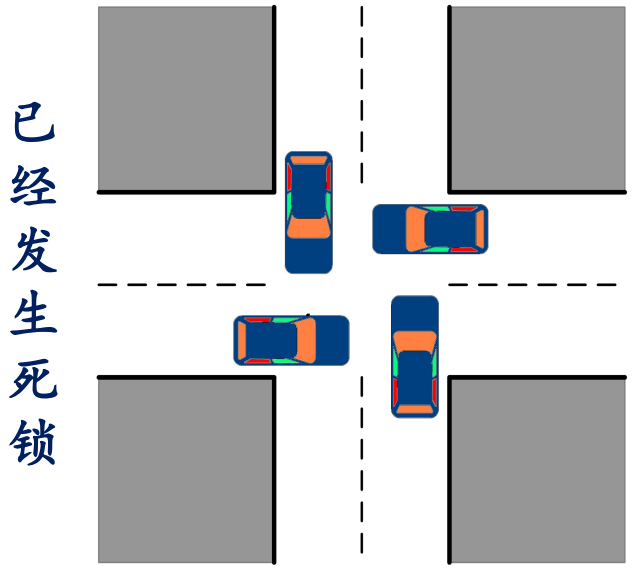
- **进程推进顺序非法**：若按曲线④的顺序推进时，进入**不安全区D内**，两进程再推进会产生死锁。

3.5.3 死锁定义、必要条件和处理方法

◆ 死锁（deadlock）的定义

- 指多个进程因**竞争共享资源**而造成的一种僵局，若无外力作用，这些进程都将永远不能再向前推进。
- **定义：**一组进程中，每个进程都无限等待被该组进程中另一进程所**占有的资源**，因而永远无法得到的资源，这种现象称为进程**死锁**，这一组进程就称为**死锁进程**。





3.5.3 死锁定义、必要条件和处理方法

◆ 基本概念

➢ 指多个进程因**竞争共享资源**而造成的一种僵局，若无外力作用，这些进程都将永远不能再向前推进。

可分配空间为200K

P1:	P2:
……	……
Request 80K bytes；	Request 70K bytes；
……	……
Request 60K bytes；	Request 80K bytes；

当两个进程都执行第二次空间请求时，发生死锁

3.5.3 死锁定义、必要条件和处理方法

◆ 相关结论

- 参与死锁的进程最少是两个
- 参与死锁的进程至少有两个已经占有资源
- 参与死锁的所有进程都在等待资源
- 参与死锁的进程是当前系统中所有进程的子集
- 如果死锁发生，会浪费大量系统资源，甚至导致系统崩溃

3.5.3 死锁定义、必要条件和处理方法

◆ 产生死锁的必要条件 （A→B）

- 死锁的发生必须同时具备下列四个必要条件
- **互斥条件：**指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由一个进程占用
- **请求和保持条件：**指进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源又已被其它进程占有，此时请求进程阻塞，但又对自己已获得的其它资源保持不放
- **不剥夺条件：**指进程已获得的资源，在未使用完之前，不能被剥夺，只能在使用完时由自己释放。
- **循环等待条件：**发生死锁时必然存在进程资源的环形链

3.5.3 死锁定义、必要条件和处理方法

◆ 处理死锁的基本方法

- **预防死锁**（较简单和直观的事先预防方法）：通过设置某些限制条件，去**破坏**产生死锁的四个**必要条件**中的一个或几个条件，来预防发生死锁
- **避免死锁**（事先预防的策略）：在资源的动态分配过程中，用某种方法去防止系统进入不安全状态

- **检测死锁**：通过系统所设置的检测机构及时地检测出死锁的发生，并精确地确定与死锁有关的进程和资源，采取适当措施，从系统中将已发生的死锁清除掉。
- **解除死锁**（与检测死锁相配套）：撤消或挂起一些进程，以便回收一些资源，再将这些资源分配给已处于阻塞状态的进程，使之转为就绪状态，以继续运行。

处理机调度与死锁

- 处理机调度的层次和调度算法的目标
- 作业与作业调度
- 进程调度
- 实时调度
- 死锁概述
- 预防死锁
- 避免死锁
- 死锁的检测与解除

处理机调度与死锁

- 破坏“请求和保持“条件
- 破坏“不可抢占”条件
- 破坏“循环等待”事件

预防死锁

避免死锁

死锁的检测与解除

3.6 预防死锁

◆ 破坏必要条件

- 在系统设计时确定资源分配算法，保证不发生死锁。具体的做法是破坏产生死锁的四个必要条件之一
- **破坏“请求和保持”条件**
  - **资源一次性分配**--破坏请求和保持条件
- **破坏“不可抢占”条件**
  - **可剥夺资源**--即当某进程新的资源未满足时，释放已占有的资源，破坏不可剥夺条件
- **破坏“循环等待”条件**
  - **资源有序分配法**--系统给每类资源赋予一个编号，每一个进程按编号递增的顺序请求资源，释放则相反，破坏环路等待条件

3.6.1 破坏“请求和保持”条件

◆ 第一种协议

- 系统要求所有进程要一次性的申请在整个运行过程所需要得全部资源，在整个运行期间，再不会提出资源要求，从而摒弃了请求条件
- 优点：简单、易于实现、安全
- 缺点：资源严重浪费，进程延迟运行，使进程经常发生饥饿现象

◆ 第二种协议

- 允许一个进程只获得允许初期所需资源后便开始运行，允许过程成再逐步释放已分配给自己、且已用毕的全部资源，然后再请求新的所需资源
- 优点：进程可更快完成任务、提高设备利用率、减少进程饥饿的机率

3.6.1 破坏“不可抢占”条件

◆ 破坏“不可抢占”条件

- 进程在需要资源时才提出请求，一个已经保持了某些资源的进程当它再提出新的资源要求而不能立即得到满足时，必须释放它已经保持的所有资源待以后需要时再申请，也就是认为资源被剥夺了
- **缺点**
  - 实施复杂、代价大
  - 反复申请和释放资源使进程的执行无限地推迟
  - 延长了进程的周转时间
  - 增加了系统开销
  - 降低了系统吞吐量

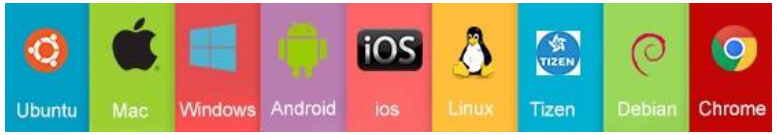
3.6.1 破坏“循环等待”条件

◆ 破坏“循环等待”条件

- 系统将所有资源按类型进行线性排队，并赋予不同的序号，所有进程对资源的请求必须严格按资源序号递增的次序提出，在资源分配图中不可能再出现环路
- **优点**
  - 资源利用率和系统吞吐量提高
- **缺点**
  - 资源序号必须相对稳定，新设备类型增加难
  - 进程使用资源的顺序与系统规定的顺序不同造成资源浪费
  - 限制用户简单、自主编程.

F(tape drive)=1;
F(disk drive)=5;
F(printer)=12;

处理机调度与死锁



- 处理机调度的层次和调度算法的目标
- 作业与作业调度
- 进程调度 ★
- 实时调度
- 死锁概述
- 预防死锁 ★
- 避免死锁
- 死锁的检测与解除

处理机调度与死锁

- 系统安全状态
- 利用银行家算法避免死锁

避免死锁——事先预防

死锁的检测与解除

3.7.1 系统的安全状态

◆ 安全状态

- 只要系统始终处于**安全状态**，便可**避免死锁**
- 系统资源分配前先**计算系统安全性**，若此次分配不会导致系统进入不安全状态，便将资源分配给进程，否则进程等待

3.7.1 系统的安全状态

◆ 安全状态定义

- 指系统按某种顺序<P1，P2，…，Pn>，来为每个进程分配其所需资源，直至最大需求，使每个进程都可顺序完成。
- **安全序列**
  - 一个进程序列<P1，P2，…，Pn>是安全的，如果对于每一个进程Pi(1≤i≤n），它以后尚需要的资源量不超过系统当前剩余资源量与所有进程Pj (j<i)当前占有资源量之和，系统处于安全状态
- 如果系统不存在**安全序列**，则称系统处于不安全状态
- 并非所有不安全状态都是死锁状态，但当系统进入不安全状态后，便可能进入死锁状态.
- 只要系统处于安全状态，便可**避免**死锁状态
- 避免死锁的**实质**是使系统**不进入不安全状态**

3.7.1 系统的安全状态

◆ 安全状态之例

- 系统有三个进程P1, P2, P3，有13台磁带机。
- P1要求9台磁带机, P2和P3分别要求4台和11台
- 设T0时刻进程P1、P2和P3分别获得2台、2台和5台，尚有4台空闲未分

进程	最大需求	已分配	可用
P_1	9	2	4
P_2	4	2	
P_3	11	5	

是否存在安全序列?

3.7.1 系统的安全状态

◆ 安全状态之例

- 系统有三个进程P1, P2, P3，有13台磁带机。
- P1要求9台磁带机, P2和P3分别要求4台和11台
- 设T0时刻进程P1、P2和P3分别获得2台、2台和5台，尚有3台空闲未分

进程	最大需求	已分配	可用
P_1	9	2	4
P_2	4	2	
P_3	11	5	

- 经分析发现：在T0时刻是安全的，因为存在安全序列<P2, P3, P1>
- 只要系统按安全序列分配资源，每个进程便可顺利完成.

3.7.1 系统的安全状态

◆ 由安全状态向不安全状态转换

- 系统有三个进程P1, P2, P3，有13台磁带机。
- P1要求9台磁带机, P2和P3分别要求4台和11台
- 设T0时刻进程P1、P2和P3分别获得2台、2台和5台，尚有3台空闲未分

进程	最大需求	已分配	可用
P_1	9	2	4
P_2	4	2	
P_3	11	5	

- 如果不按照安全序列<P2, P3, P1>分配资源，则系统可能会由安全状态向不安全状态转换

3.7.1 系统的安全状态

◆ 由安全状态向不安全状态转换

- 系统有三个进程P1, P2, P3，有13台磁带机。
- P1要求9台磁带机, P2和P3分别要求4台和11台
- 设T0时刻进程P1、P2和P3分别获得2台、2台和5台，尚有3台空闲未分
- 在T0时刻以后，P1又请求1台磁带机，若此时系统把剩余4台中的1台分配给P1。把其余的2台分配给P2，P2完成后释放出5台，不能满足P1和P3均尚需6台的要求，致使它们彼此都在等待对方释放资源——死锁

进程	最大需求	已分配	可用
P_1	9	3	3
P_2	4	2	
P_3	11	5	

3.7.1 系统的安全状态

◆ 由安全状态向不安全状态转换

- 系统有三个进程P1, P2, P3，有13台磁带机。
- P1要求9台磁带机, P2和P3分别要求4台和11台
- 设T0时刻进程P1、P2和P3分别获得2台、2台和5台，尚有3台空闲未分
- 在T0时刻以后，P1又请求1台磁带机，若此时系统把剩余4台中的1台分配给P1。把其余的2台分配给P2，P2完成后释放出5台，不能满足P1和P3均尚需6台的要求，致使它们彼此都在等待对方释放资源——死锁

避免死锁基本思想： 确保系统始终处于安全状态。一个系统开始时是处于安全状态的，当有进程请求一个可用资源时，系统需要对该进程的请求进行计算，若资源分配给进程后系统仍处于安全状态，才将资源分配给进程

3.7.2 利用银行家算法避免死锁

- **银行家算法中的数据结构**
  - **Available：可利用资源向量**
  - **Max：最大需求矩阵**
  - **Allocation：分配矩阵**
  - **Need：需求矩阵**

3.7.2 利用银行家算法避免死锁

◆ 银行家算法中的数据结构

➢ Available：可利用资源向量

- 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Rj类资源K个

3.7.2 利用银行家算法避免死锁

◆ 银行家算法中的数据结构

➢ Max：最大需求矩阵

- 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Rj类资源的最大数目为K

3.7.2 利用银行家算法避免死锁

◆ 银行家算法中的数据结构

➢ Allocation：分配矩阵

- 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Rj类资源的数目为K

3.7.2 利用银行家算法避免死锁

◆ 银行家算法中的数据结构

➢ Need：需求矩阵

- 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Rj类资源K个，方能完成其任务

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

3.7.2 利用银行家算法避免死锁

◆ 银行家算法

- 假设Requesti是进程Pi的请求向量
- 如果Requesti［j］=K，表示进程Pi需要K个Rj类型的资源。
- 当Pi发出资源请求后，系统按下述步骤进行检查：

3.7.2 利用银行家算法避免死锁

◆ **银行家算法**

Available：可利用资源向量
Max：最大需求矩阵

Allocation：分配矩阵
Need：需求矩阵

➢ **步骤一：**

- 如果Requesti［j］≤Need［i,j］，便转向步骤二；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。

➢ **步骤二：**

- 如果Requesti［j］≤Available［j］，便转向步骤三；否则， 表示尚无足够资源，Pi须等待。

➢ **步骤三：**

- 系统试探着把资源分配给进程Pi，并修改下面数据结构中的数值：
- Available［j］：=Available［j］-Requesti［j］；
- Allocation［i,j］：=Allocation［i,j］+Requesti［j］
- Need［i,j］：=Need［i,j］-Requesti［j］

➢ **步骤四：**

- 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Pi，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Pi等待。

3.7.2 利用银行家算法避免死锁

- **银行家算法调用安全算法**
  - **步骤一：设置两个变量**
    - **工作向量Work:**
      - 它表示系统可提供给进程继续运行所需的各类资源数目，它含有m个元素
      - 在执行安全算法开始时，Work：=Available;
    - **Finish:**
      - 它表示系统是否有足够的资源分配给进程，使之运行完成。
      - 开始时先做Finish［i］：=false; 当有足够资源分配给进程时， 再令Finish［i］：=true。

3.7.2 利用银行家算法避免死锁

◆ 银行家算法调用安全算法

- **步骤二：**
  - 从进程集合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，执行步骤三，否则，执行步骤四；
- **步骤三：**
  - 当进程Pi获得资源后，可顺利执行，直至完成，并释放出分配给它的资源，故应执行：
  - Work［j］：＝Work［i］+Allocation［i,j］；
  - Finish［i］：＝true;
  - go to step 2;
- **步骤四：**
  - 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。

3.7.2 利用银行家算法避免死锁

◆ 银行家算法之例

- 假定系统中有五个进程：

  {P0, P1, P2, P3, P4}

- 三类资源：

  {A, B, C} → 10, 5, 7

- 在T0时刻的资源分配情况如图所示：

3.7.2 利用银行家算法避免死锁

{A, B, C} → 10, 5, 7

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

T_0时刻的安全性?

3.7.2 利用银行家算法避免死锁

T_0时刻的安全性

资源 / 进程	Work A B C	Need A B C	Allocation A B C	Work+ Allocation A B 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

3.7.2 利用银行家算法避免死锁

T_0时刻的安全性

资源 / 进程	Work A B C	Need A B C	Allocation A B C	Work+ Allocation A B 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_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

T0时刻是安全的
存在安全序列：P1→P3 →P4 →P2 →P0

3.7.2 利用银行家算法避免死锁

◆ **银行家算法之例**

- **P1请求资源：**
- **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)≤Available1(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

3.7.2 利用银行家算法避免死锁

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

3.7.2 利用银行家算法避免死锁

资源 / 进程	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

3.7.2 利用银行家算法避免死锁

资源 / 进程	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_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请求可满足
分配后，存在安全序列：$P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$

3.7.2 利用银行家算法避免死锁

◆ 银行家算法之例

- **P4请求资源：**
- **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。

3.7.2 利用银行家算法避免死锁

◆ 银行家算法之例

- P0请求资源：
- 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)≤Need (7, 4, 3)

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

请计算：P_0请求是否能够满足?

如果是，给出安全序列：$P_?$ →$P_?$ →$P_?$ →$P_?$ →$P_?$

3.7.2 利用银行家算法避免死锁

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

3.7.2 利用银行家算法避免死锁

资源 / 进程	Max A B C	Allocation A B C	Need A B C	Available A B C
P_0	7 5 3	0 1 0	7 4 3	
P_4	4 3 3	0 0 2	4 3 1	

Available(2,1,0)不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。

3.7.2 利用银行家算法避免死锁

◆ 银行家算法之例

- **P0请求资源：**
- **发出请求向量Requst0(0，1，0)，系统按银行家算法进行检查：**

① Request0(0, 1, 0) ≤ Need (7, 4, 3)

② Request0(0, 1, 0) ≤ Available(2, 3, 0)

请计算：P_0请求是否能够满足?
如果是，给出安全序列：$P_?$ →$P_?$ →$P_?$ →$P_?$ →$P_?$

3.7.2 利用银行家算法避免死锁

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

3.7.2 利用银行家算法避免死锁

资源 / 进程	Work A B C	Need A B C	Allocation A B C	Work+ Allocation A B C	Finish
P_1	2 2 0	0 2 0	3 0 2	5 2 2	True
P_3	5 2 2	0 1 1	2 1 1	7 3 3	True
P_4	7 3 3	4 3 1	0 0 2	7 3 5	True
P_0	7 3 5	7 3 3	0 2 0	7 5 5	True
P_2	7 5 5	6 0 0	3 0 2	10 5 7	true

3.7.2 利用银行家算法避免死锁

资源 / 进程	Work A B C	Need A B C	Allocation A B C	Work+ Allocation A B C	Finish
P_1	2 2 0	0 2 0	3 0 2	5 2 2	True
P_3	5 2 2	0 1 1	2 1 1	7 3 3	True
P_4	7 3 3	4 3 1	0 0 2	7 3 5	True
P_0	7 3 5	7 3 3	0 2 0	7 5 5	True
P_2	7 5 5	6 0 0	3 0 2	10 5 7	true

P_0请求Requst0(0，1，0)可满足，分配后存在安全序列：$P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$

3.7.2 利用银行家算法避免死锁

◆ 银行家算法举例

- 设系统中有3种类型的资源(A、B、C)和5个进程(P1、P2、P3、P4、P5)。
- A资源的总量为17，B资源的总量为5，C资源的总量为20。
- 在T0时刻系统状态如下表所示，系统采用银行家算法实施死锁避免策略

进程	已经分配资源 (Allocation)			最大需求矩阵 (Claim)		
	A	B	C	A	B	C
P1	2	1	2	5	5	9
P2	4	0	2	5	3	6
P3	4	0	5	4	0	11
P4	2	0	4	4	2	5
P5	3	1	4	4	2	4

3.7.2 利用银行家算法避免死锁

◆ 银行家算法举例

- 设系统中有3种类型的资源(A、B、C)和5个进程(P1、P2、P3、P4、P5)。
- A资源的总量为17，B资源的总量为5，C资源的总量为20。
- 在T0时刻系统状态如下表所示，系统采用银行家算法实施死锁避免策略

进程	已经分配资源 (Allocation)			最大需求矩阵 (Claim)		
	A	B	C	A	B	C
P1	2	1	2	5	5	9
P2	4	0	2	5	3	6
P3	4	0	5	4	0	11
P4	2	0	4	4	2	5
P5	3	1	4	4	2	4

(1) T0时刻的各资源剩余数量为多少?T0时刻的是否为安全状态? 若是，请给出其中可能的一种安全序列，并依照该序列，写出各资源的回收步骤。

3.7.2 利

T0时刻的各资源剩余数量为多少?
A资源的总量为17，B资源的总量为5，C资源的总量为20

资源 / 进程	Max A B C	Allocation A B C	Need A B C	Available A B C
P_1	5 5 9	2 1 2	3 4 7	
P_2	5 3 6	4 0 2	1 3 4	
P_3	4 0 11	4 0 5	0 0 6	2 3 3
P_4	4 2 5	2 0 4	2 2 1	
P_5	4 2 4	3 1 4	1 1 0	
		15 2 17		

3.7.2 和

T0时刻的是否为安全状态? 若是，请给出其中可能的一种安全序列，并依照该序列，写出各资源的回收步骤

资源 / 进程	Work A B C	Need A B C	Allocation A B C	Work+ Allocation A B C	Finish
P_4	2 3 3	2 2 1	2 0 4	4 3 7	True
P_2	4 3 7	1 3 4	4 0 2	8 3 9	True
P_3	8 3 9	0 0 6	4 0 5	12 3 14	True
P_5	12 3 14	1 1 0	3 1 4	15 4 18	True
P_1	15 4 18	3 4 7	2 1 2	17 5 20	true

3.7.2 利

T0时刻的是否为安全状态? 若是，请给出其中可能的一种安全序列，并依照该序列，写出各资源的回收步骤

资源 / 进程	Work A B C	Need A B C	Allocation A B C	Work+ Allocation A B C	Finish
P_4	2 3 3	2 2 1	2 0 4	4 3 7	True
P_2	4 3 7	1 3 4	4 0 2	8 3 9	True
P_5	12 3 14	1 1 0	3 1 4	15 4 18	True
P_1	15 4 18	3 4 7	2 1 2	17 5 20	true

T0时刻是安全的

存在安全序列：P4→P2 →P3 →P5 →P1

3.7.2 利用银行家算法避免死锁

◆ 银行家算法举例

- 设系统中有3种类型的资源(A、B、C)和5个进程(P1、P2、P3、P4、P5)。
- A资源的总量为17，B资源的总量为5，C资源的总量为20。
- 在T0时刻系统状态如下表所示，系统采用银行家算法实施死锁避免策略

进程	已经分配资源 (Allocation)			最大需求矩阵 (Claim)		
	A	B	C	A	B	C
P1	2	1	2	5	5	9
P2	4	0	2	5	3	6
P3	4	0	5	4	0	11
P4	2	0	4	4	2	5
P5	3	1	4	4	2	4

在T0时刻，如果进程P1继续对ABC三类资源提出请求Request (2, 2, 2)后，系统能否将资源分配给P1进程?

3.7.2 利用银行家算法避免死锁

在T0时刻，如果进程P1继续对ABC三类资源提出请求Request (2, 2, 2)后，系统能否将资源分配给P1进程?

◆ **银行家算法之例**

- **P1请求资源：**
- **发出请求向量Requst1(2，2，2)，系统按银行家算法进行检查：**

① Requst1(2，2，2) ≤Need (3, 4, 7)

② Requst1(2，2，2) ≤Available(2, 3, 3)

3.7.2 利

在T0时刻，如果进程P1继续对ABC三类资源提出请求Request (2, 2, 2)后，系统能否将资源分配给P1进程?

资源 / 进程	Max A B C	Allocation A B C	Need A B C	Available A B C
P_1	5 5 9	2 1 2 (4 3 4)	3 4 7 (1 2 5)	
P_2	5 3 6	4 0 2	1 3 4	
P_3	4 0 11	4 0 5	0 0 6	2 3 3 (0 1 1)
P_4	4 2 5	2 0 4	2 2 1	
P_5	4 2 4	3 1 4	1 1 0	

3.7.2 利用银

在T0时刻，如果进程P1继续对ABC三类资源提出请求Request (2, 2, 2)后，系统能否将资源分配给P1进程?

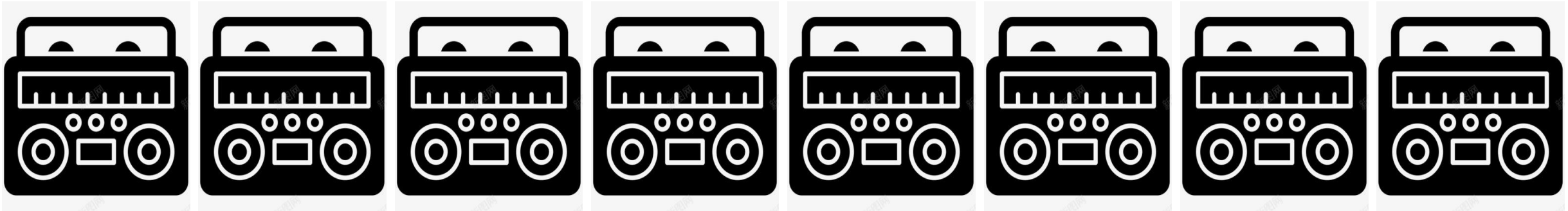
资源 / 进程	Max A B C	Allocation A B C	Need A B C	Available A B C
P_1	5 5 9	2 1 2	3 4 7	
P_4	4 2 5	2 0 4	2 2 1	
P_5	4 2 4	3 1 4	1 1 0	

Available(0,1,0)不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

3.8.2 死锁部分例题

◆一台计算机有八台磁带机。它们由N个进程竞争使用，每个进程可能需要3台磁带机。请问N为多少时，系统没有死锁危险，并说明其原因。



◆ N = 2 永远不会

◆ N = 3，3个进程竞争8台设备，也不会死锁

3.8.2 死锁部分例题

◆**N=4时**：存在安全序列<P1,P2,P3,P4>，按照安全序列推进时不会死锁，否则存在死锁危险.

	P1	P2	P3	P4	8台
Max	3	3	3	3	
Allocation	1	1	1	1	剩4台

◆**N=5时**：存在安全序列<P1,P2,P3,P4,P5>，按照安全序列推进时不会死锁，否则存在死锁危险.

	P1	P2	P3	P4	P5	8台
Max	3	3	3	3	3	
Allocation	1	1	1	1	1	剩3台

3.8.2 死锁部分例题

◆**N=6时**：存在安全序列<P1,P2,P3,P4,P5,P6>按照安全序列推进时不会死锁，否则存在死锁危险.

	P1	P2	P3	P4	P5	P6	8台
Max	3	3	3	3	3	3	
Allocation	1	1	1	1	1	1	剩2台

◆**N=7时**：不存在安全序列，会死锁

	P1	P2	P3	P4	P5	P6	P7	8台
Max	3	3	3	3	3	3	3	
Allocation	1	1	1	1	1	1	1	剩1台

3.8.2 死锁部分例题

◆N个进程共享某种资源r，该资源共有m个可分配单位，每个进程每次一个地申请或释放资源单位，假设每个进程对该资源的最大需求量均小于m，且各进程的最大需求量之和小于m+n，试证明这个系统中不可能发生死锁。

设max（i）表示第I个进程的最大需求量，need（I）表示第I个进程还需要的资源量。Alloc（I）表示第I个进程已分配的资源量，可知：

$$max(1)+\ldots+max(n) < m+n$$

$$(need(1)+\ldots+need(n))+(alloc(1)+\ldots+alloc(n)) < m+n$$

3.8.2 死锁部分例题

- 设max（i）表示第I个进程的最大需求量，need（I）表示第I个进程还需要的资源量。Alloc（I）表示第I个进程已分配的资源量，可知：

$$max(1)+\ldots+max(n) < m+n$$

$$(need(1)+\ldots+need(n))+(alloc(1)+\ldots+alloc(n)) < m+n$$

- 如果在这个系统中发生了死锁，一方面m个资源应该全部分配出去，即：

$$alloc(1)+\ldots+alloc(n)=m$$

- 另一方面所有进程将陷入无限等待的状态，由上述两式可得 ：

$$need(1)+\ldots+need(n)<n$$

- 上式表示发生死锁后，n个进程还需要的最小资源量之和小于n，这意味着此刻至少存在着一个进程I，need（I）=0，即它已获得了所需的全部资源。既然该进程已获得了所需的全部资源，那么它就能执行完成并释放它所占有的全部资源，这与前面的假设矛盾，因此这个系统中不可能发生死锁。

处理机调度与死锁

- 处理机调度的层次和调度算法的目标
- 作业与作业调度
- 进程调度
- 实时调度
- 死锁概述
- 预防死锁
- 避免死锁
- 死锁的检测与解除


3.8 死锁的检测与解除

◆如果再系统中，既不采取死锁预防措施，也未配有死锁避免算法，则很可能发生死锁，系统提供两个算法

- **死锁检测算法**：检测系统状态，确定系统是否发生死锁
- **死锁解除算法**：将系统从死锁状态中解脱出来

死锁的检测

- 资源分配图
- 死锁定理
- 死锁检测中的数据结构

死锁的解除

- 利用资源抢占
- 利用回退释放资源
- 利用结束死锁进程释放资源

3.8.1 死锁的检测

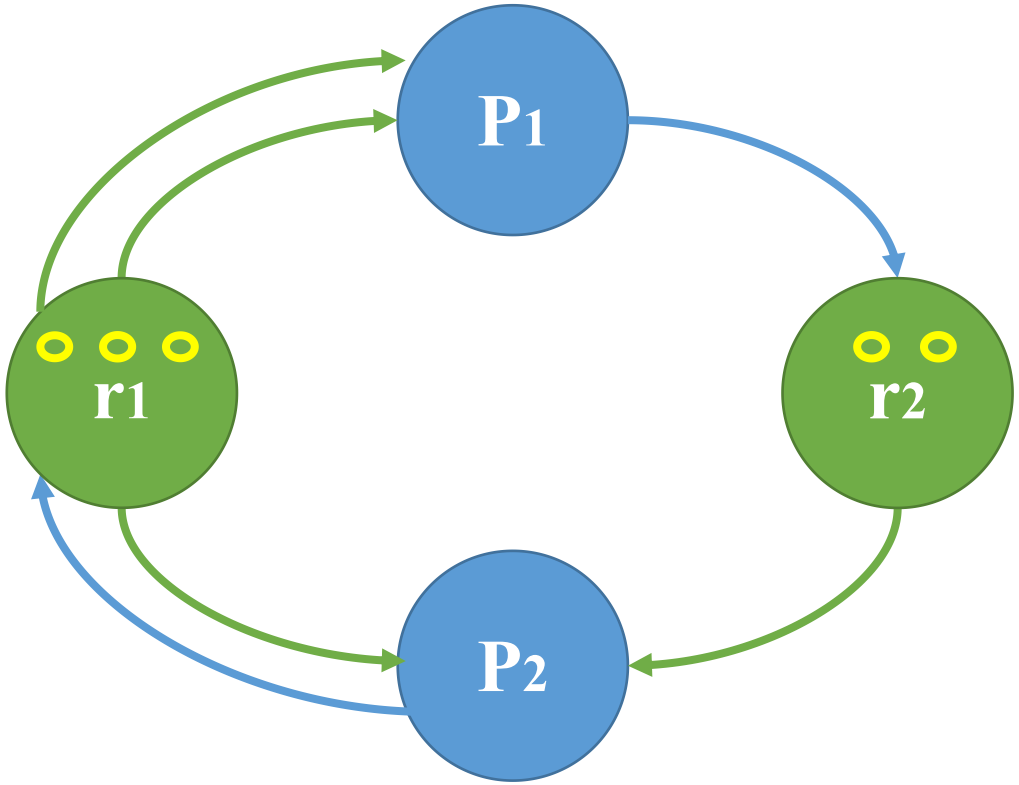
◆ 死锁的检测

- ➢ 为了能对系统中是否发生了死锁进行检测，在系统中必须：
  - 保存有关资源的请求和分配信息；
  - 提供一种算法，以利用这些信息来检测系统是否已进入死锁状态

◆资源分配图

- ➢ 该图是由一组结点N和一组边E所组成的一个对偶G=(N，E)，它具有下述形式的定义和限制：：
  - 把N分为两个互斥的子集，即：一组进程结点P={p1，p2，…，pn}，一组资源结点R={r1，r2，…，rn}，N=P∪R，如：P={p1，p2}，R={r1，r2}，N={r1，r2}∪{p1，p2}。

3.8.1 死锁的检测



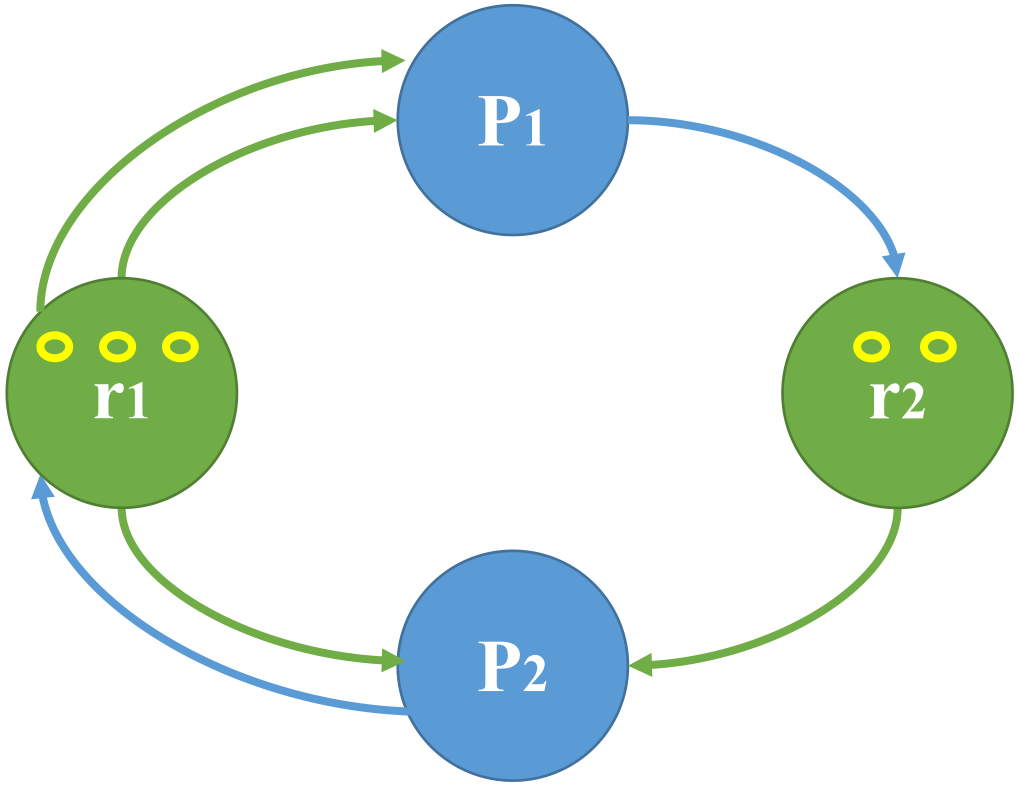
- 把N分为两个互斥的子集，即：一组进程结点P={p1，p2，…，pn}，一组资源结点R={r1，r2，…，rn}，N=P∪R，如：P={p1，p2}，R={r1，r2}，N={r1，r2}∪{p1，p2}。

3.8.1 死锁的检测

◆资源分配图

- 该图是由一组结点N和一组边E所组成的一个对偶G=(N，E)，它具有下述形式的定义和限制：：
  - 凡属于E中的一个边e∈E，都连接着P中的一个结点和R中的一个结点。
  - e={pi，rj}是资源请求边，由进程pi指向资源rj，它表示进程pi请求一个单位的rj资源。
  - e={rj，pi}是资源分配边，由资源rj指向进程pi，它表示把一个单位的资源rj分配给进程pi。
  - P1分得2个r1，请求1个r2; P2分得1个r1和r2，请求1个r1;

3.8.1 死锁的检测

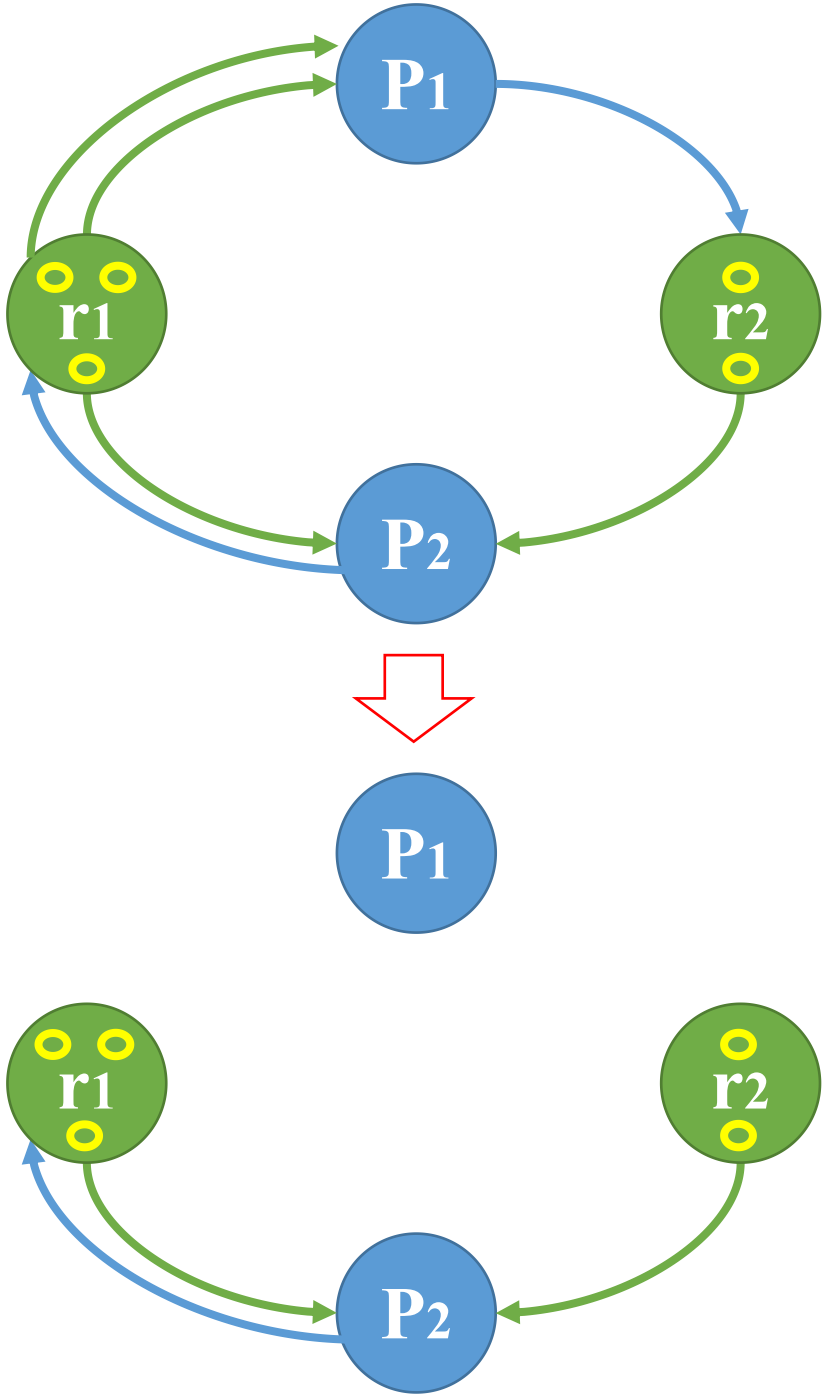


- P1分得2个r1，请求1个r2; P2分得1个r1和r2，请求1个r1;

3.8.1 死锁的检测

◆ 死锁定理

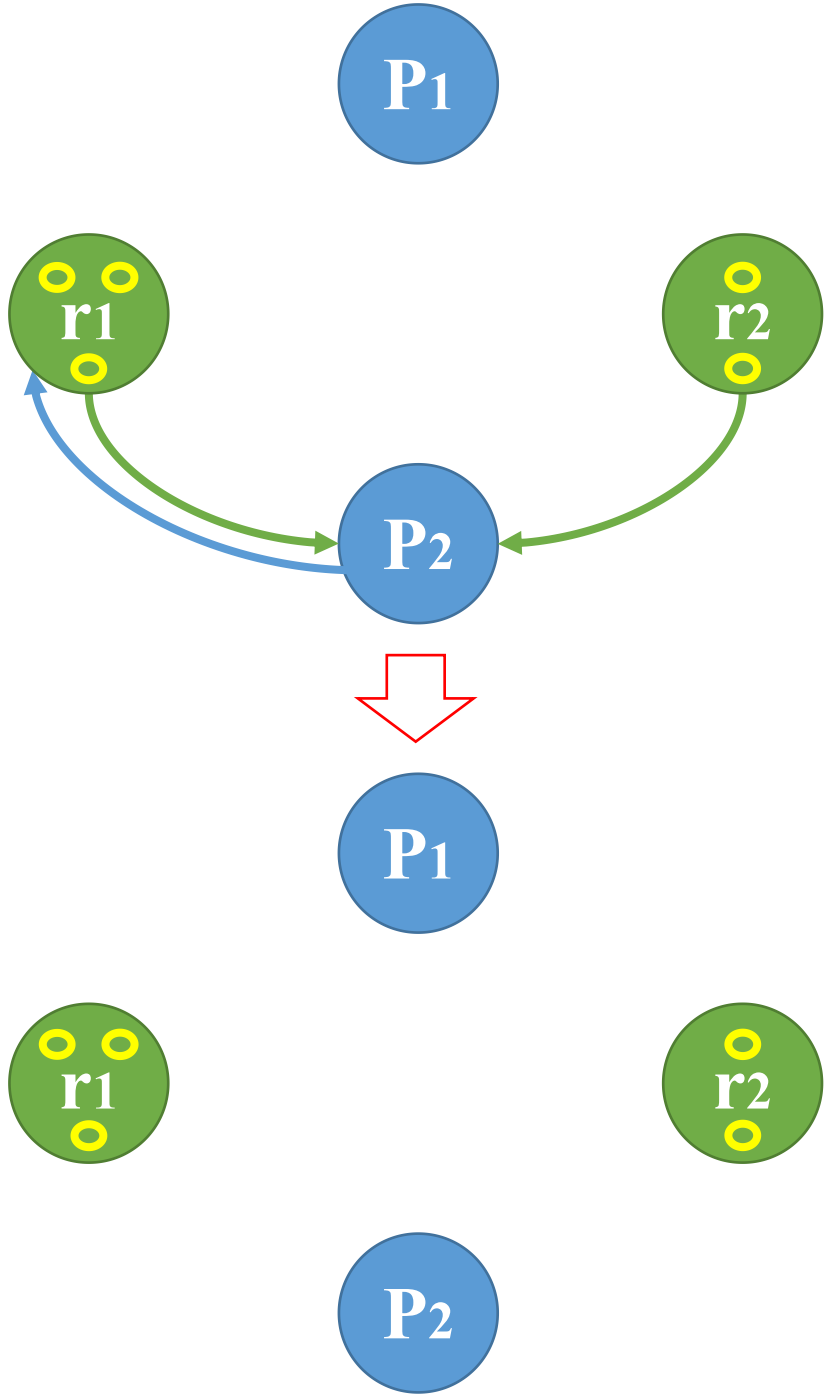
- **利用资源分配图来检测系统是否已进入死锁状态：**
  - 在资源分配图中，找出一个既不阻塞又非独立的进程结点Pi。在顺利的情况下Pi可获得所需资源而继续运行，直至运行完毕，再释放其所占有的全部资源，相当于消去pi请求边和分配边，使之成为孤立结点
  - 将p1的分配边和请求边消去：



3.8.1 死锁的检测

◆ 死锁定理

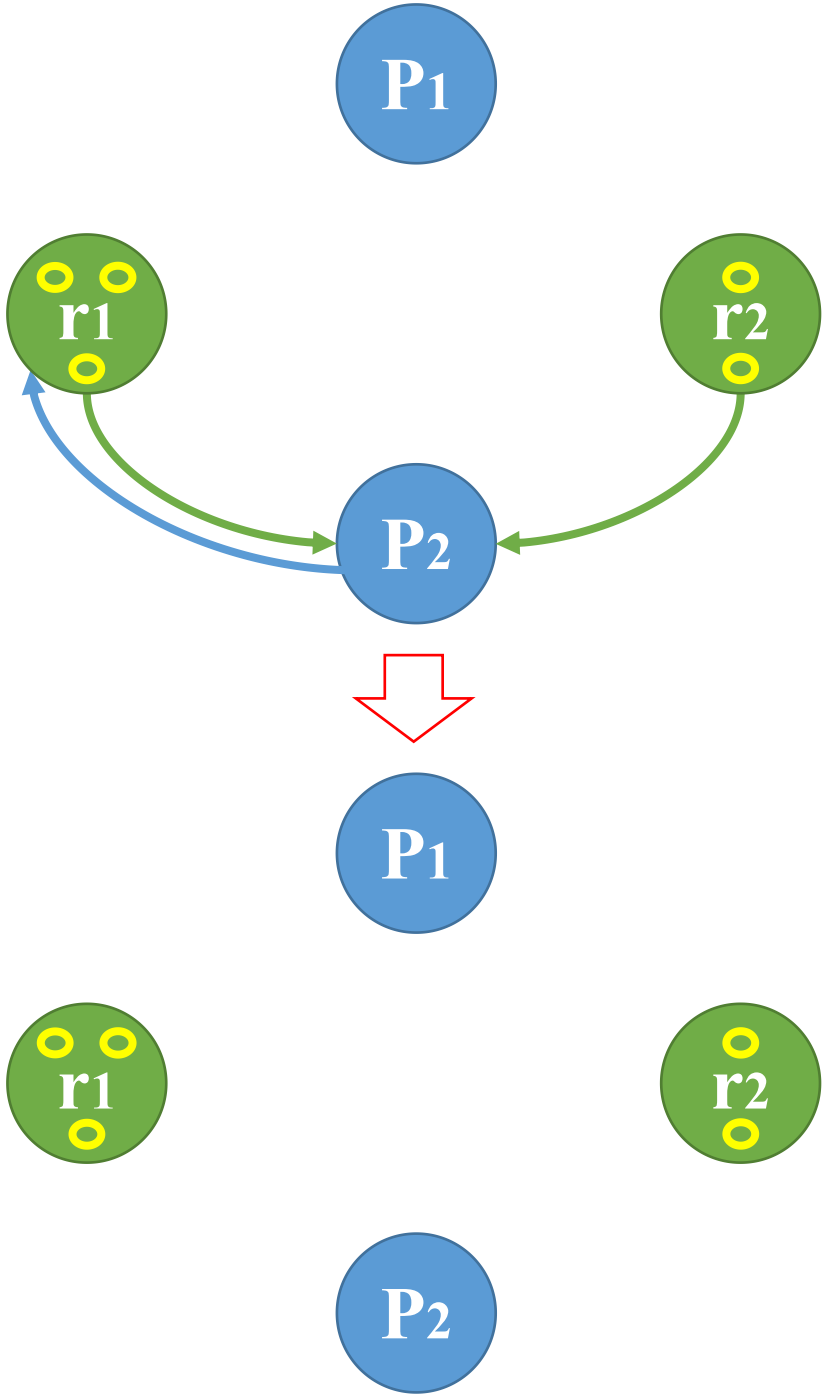
- **利用资源分配图来检测系统是否已进入死锁状态：**
  - p1释放资源后，便可使 p2获得资源而继续运行，直至p2完成后又释放出它所占有的全部资源
  - 在进行一系列的简化后，若能消去图中所有的边，使所有的进程结点都成为孤立结点，则称该图是可完全简化的；若不能通过任何过程使该图完全简化，则称该图是不可完全简化的



3.8.1 死锁的检测

◆ 死锁定理

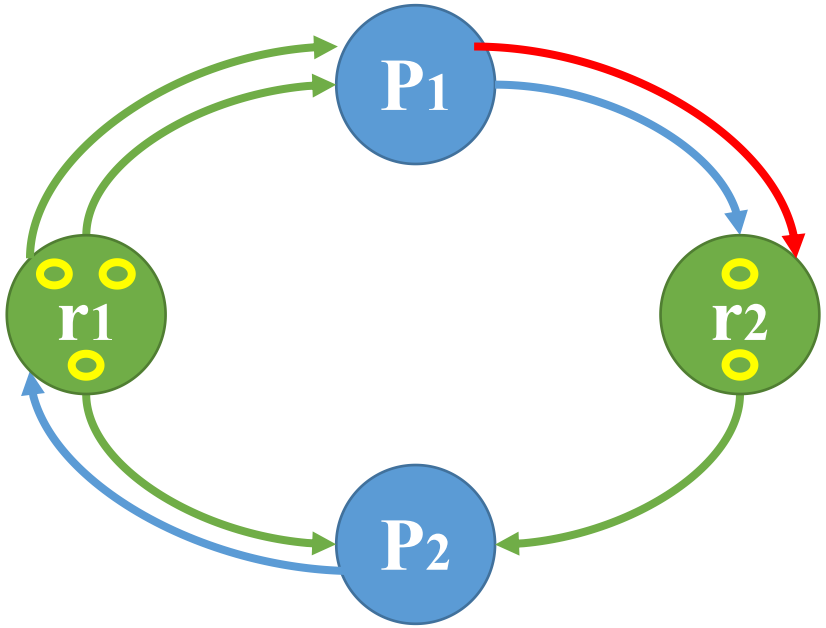
系统为死锁状态的充分条件是：当且仅当描述系统资源分配情况的资源分配图是**不能完全简化**的。该**充分条件**称为**死锁定理**。



3.8.1 死锁的检测

◆ 死锁定理

系统为死锁状态的充分条件是：当且仅当描述系统资源分配情况的资源分配图是不能完全简化的。该充分条件称为死锁定理。



3.8.1 死锁的检测

◆ 死锁检测中的数据结构

➢ 死锁检测中的数据结构类似于银行家算法中的数据结构

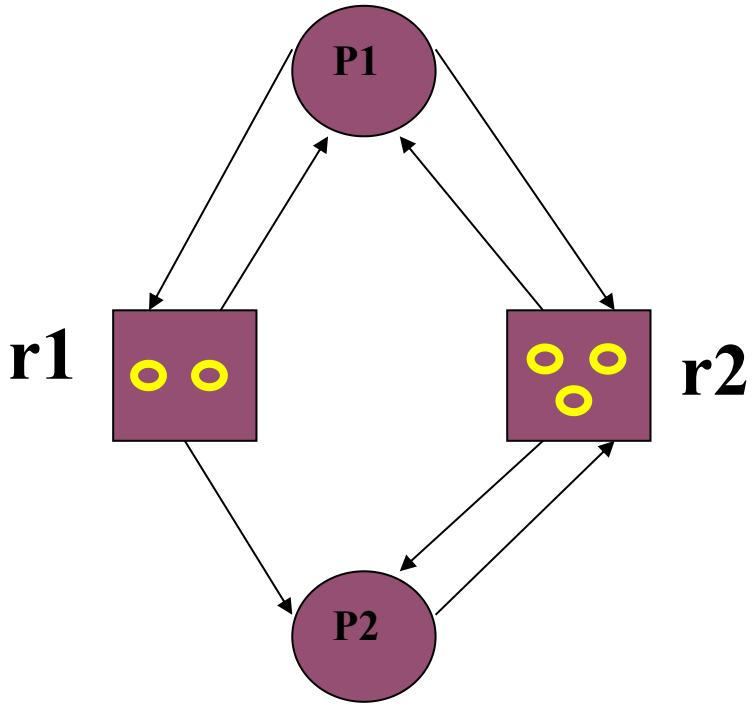
- ➢ 可利用资源向量Available，它表示了 m 类资源中每一类资源的可用数目；
- ➢ 把不占用资源的进程(向量 Allocationi:=0)记入L表中；
- ➢ 从进程集合中找到Requesti<=Work的进程，并做相应处理：
  - 将其资源分配图简化，释放出资源，增加工作向量 Work:=Work + Allocation I
  - 将它记入 L 表中
- ➢ 若不能把所有进程都记入L表中，便表明系统状态S的资源分配图是不可完全简化的。因此，系统状态将发生死锁。

3.8.2 死锁的解除

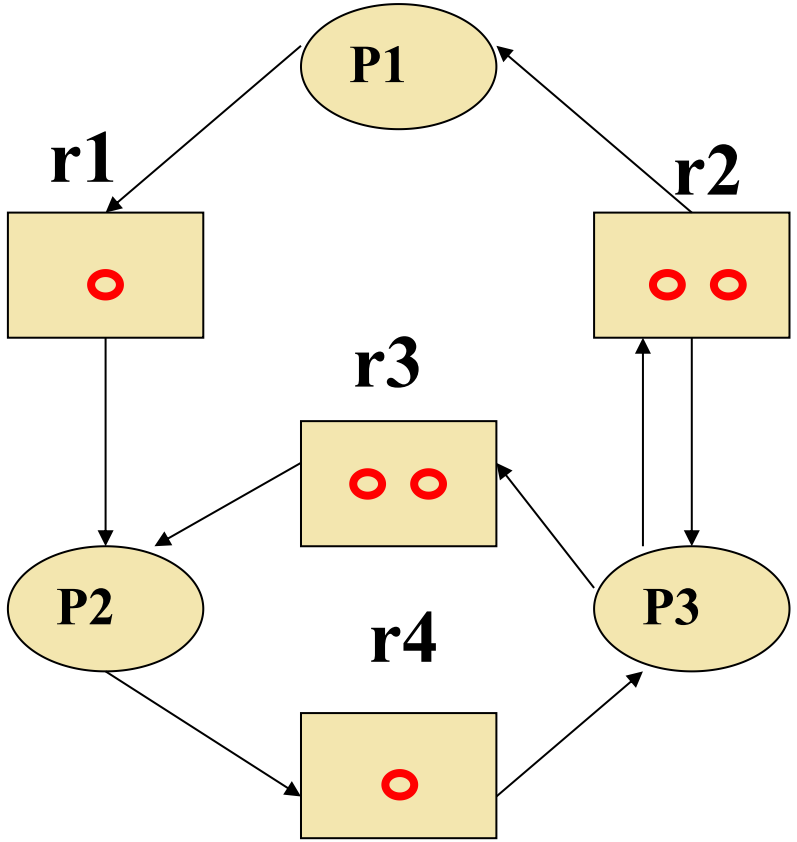
- **要从死锁状态中解脱出来，可采用方法：**
  - **抢占/剥夺资源：剥夺陷于死锁的进程占用的资源并重新分配，但不撤销该进程，直至死锁解除**
  - **终止/撤消进程：撤销陷于死锁的所有进程，或逐个撤销陷于死锁的进程，回收其资源并重新分配，直至死锁解除**
  - **回退释放资源：根据系统保存的检查点，使所有进程回退，直至解除死锁**
  - **重启计算机操作系统**

3.8.2 死锁部分例题

◆化简下图中的进程-资源图并利用死锁定理给出相应的结论



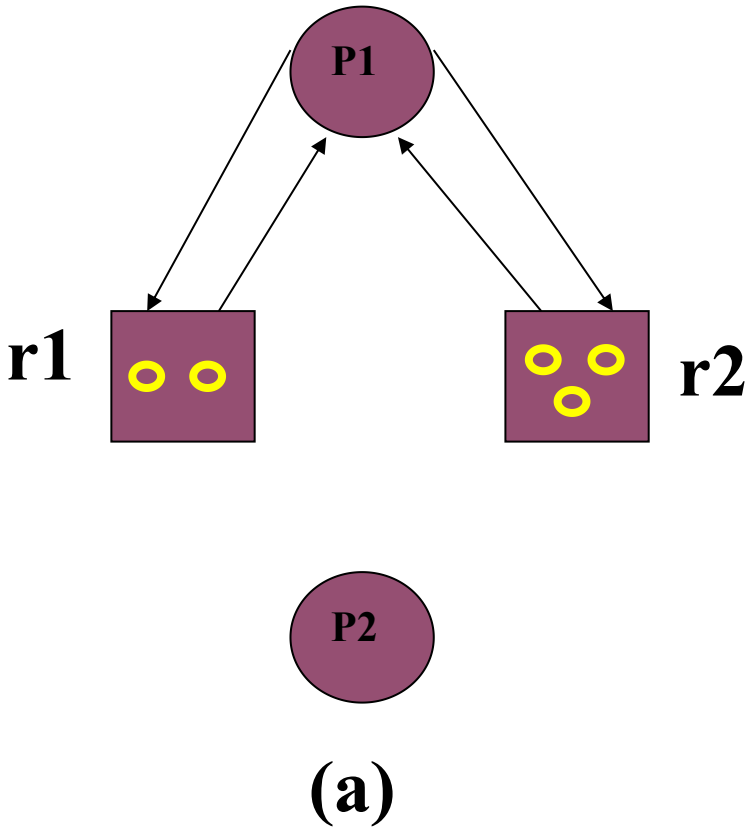
(a)



(b)

3.8.2 死锁部分例题

◆化简下图中的进程-资源图并利用死锁定理给出相应的结论

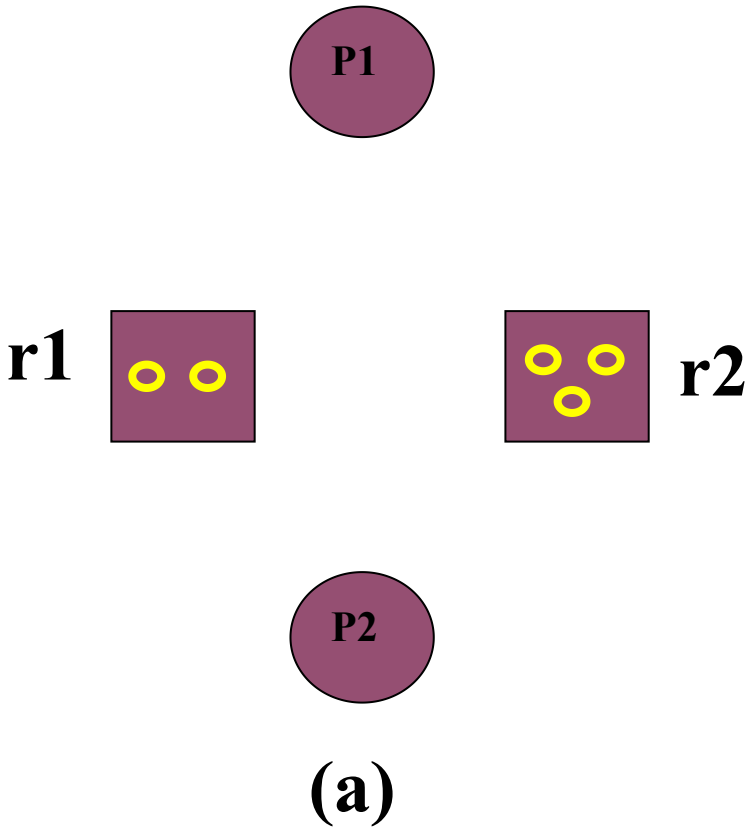


(a)

- 系统共有r1资源2个，r2类资源3个，在当前状态下只有1个r2类资源空闲；进程p2占有1个r1类资源及一个r2类资源，并申请1个r2类资源；进程p1占有1个r1类资源和1个r2类资源，并申请1个r1类资源和1个r2类资源
- 因此进程p2既是一个既不孤立又非阻塞的进程，消去进程p2的资源请求边和资源分配边；

3.8.2 死锁部分例题

◆化简下图中的进程-资源图并利用死锁定理给出相应的结论

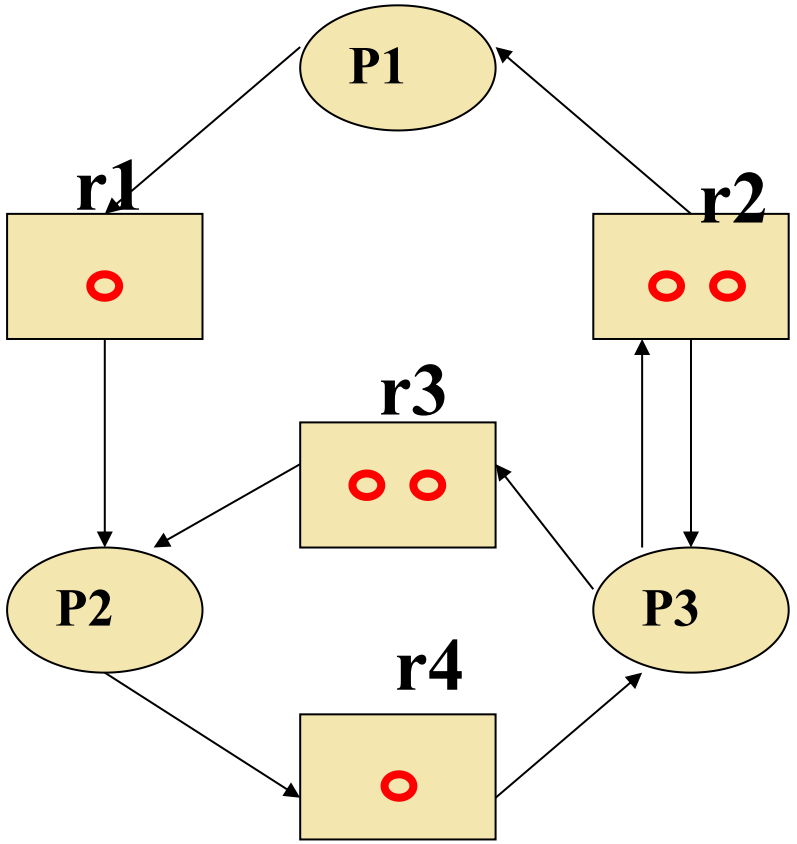


(a)

- 系统共有r1资源2个，r2类资源3个，在当前状态下只有1个r2类资源空闲；进程p2占有1个r1类资源及一个r2类资源，并申请1个r2类资源；进程p1占有1个r1类资源和1个r2类资源，并申请1个r1类资源和1个r2类资源
- 因此进程p2既是一个既不孤立又非阻塞的进程，消去进程p2的资源请求边和资源分配边；
- 当进程p2释放资源后，系统中有2个r2类空闲资源，因此系统能满足进程p1的资源申请，使得进程p1成为一个既不独立又非阻塞的资源进程，消去进程p1 的资源请求边和资源分配边；
- 由死锁定理可知：图（a）中的进程资源图不会产生死锁

3.8.2 死锁部分例题

◆化简下图中的进程-资源图并利用死锁定理给出相应的结论



(b)

- 系统有r1类资源1个，r2类资源2个，r3类资源2个，r4类资源1个。
- 在当前状态下只有1个r3类资源空闲
- 进程p1占有1个r2资源，并申请一个r1类资源；
- 进程p2占有一个r1资源和一个r3类资源，并申请一个r4类资源；
- 进程p3占有一个r4资源和一 个r2资源，并申请一个r3类资源和一个r2类资源；因此系统中的3个资源均无法向前推进；
- 由死锁定理可知，图（b）中的进程资源图会产生死锁


THEORY
PRACTICAL