操作系统

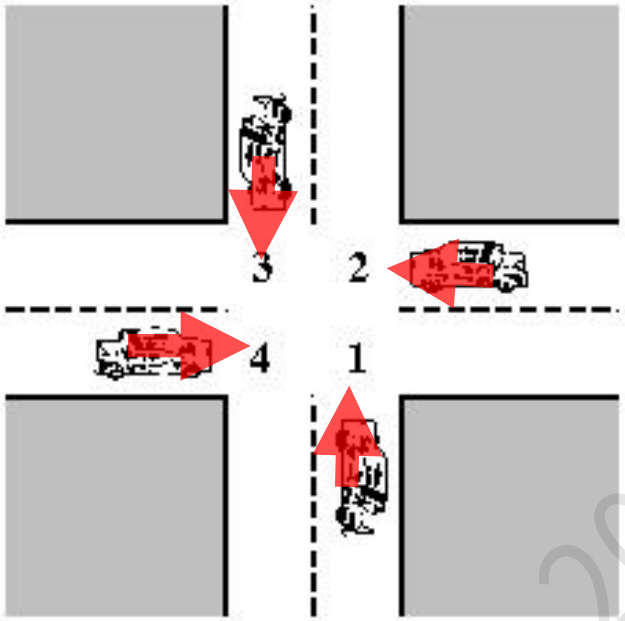
第3章5-8 死锁

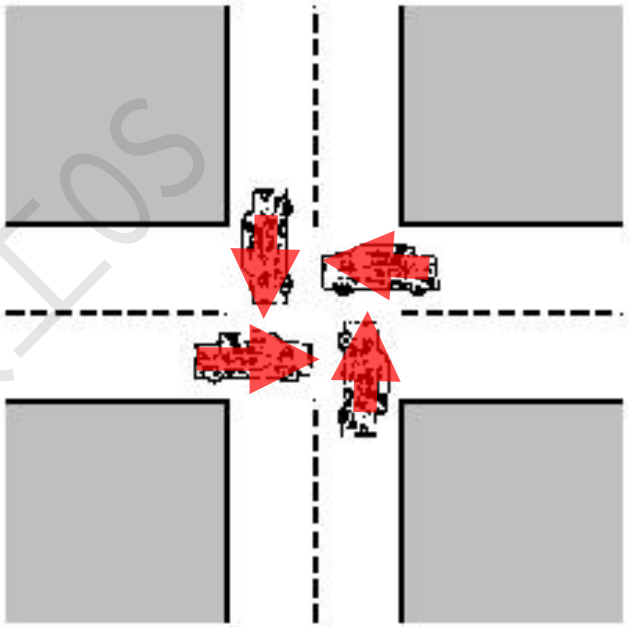
朱小军, 教授
https://xzhu.info
南京航空航天大学
计算机科学与技术学院
2025年春

生活中的死锁

交通中的死锁

交通中的死锁



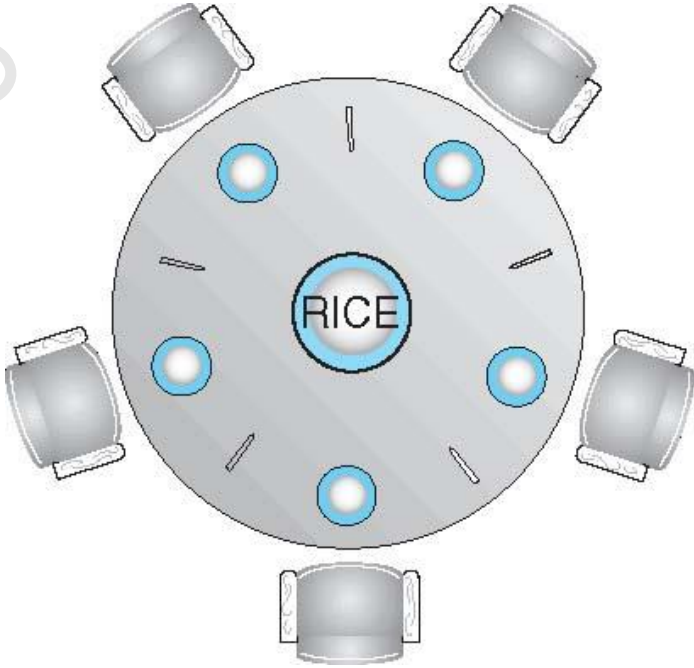


注意两个特征：

- “死”：都在等，而不是积极推进
- “锁”：推不动，锁住了，没有外部助力，永远解决不了

哲学家进餐问题

- 哲学家的一生都在两种状态中：吃或者思考
- 当他吃饭时，需要拿起两根筷子，但一次只能拿一只
- 吃完后放回筷子，继续思考
- 哲学家之间不允许抢筷子



某个时刻，每个哲学家手里拿了一根筷子，都在等另一根

目录

死锁

操作系统中的死锁

- 一组进程处于死锁状态，如果它们中每个进程都在等待只有组内其他进程才能引发的事件
  - A set of processes is in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set.

阻塞状态

死锁的必要条件

- 互斥条件
  - 资源只能由一个进程使用
- 请求和保持
  - 需要其他资源时不（完全）释放自己的资源
- 不可抢占
  - 只能由自己主动释放
- 循环等待
  - 可以找到一个进程-资源的循环链

使用场景本身的性质

运行状态相关

处理死锁的策略

- 预防或避免策略：不可能死锁
  - 采取某种协议，使得系统永远不可能进入死锁状态
- 检测和恢复策略：可能死锁->恢复
  - 允许系统进入死锁状态，然后检测死锁并进行恢复
- 忽略策略：“我相信没有死锁”
  - 假装死锁永远不会发生，不做任何处理
  - 主流操作系统，Linux 和 Windows，采用了此策略

目录

破坏必要条件

- 互斥条件
- 请求和保持
- 不可抢占
- 循环等待

预防死锁：破坏互斥条件

- 用其他方式保护共享资源
  - 找一个代理进程管理共享资源的访问，需要资源时向代理发送请求
- 仔细斟酌哪些资源必须互斥访问
  - 是否可以切割出可以共享的部分，比如，只读文件
  - 直接分配，避免共享，比如，物理内存分块后静态分给进程

预防死锁：破坏“请求和保持”条件

- 法1：一次请求所有资源，后面不请求
- 法2：请求新资源时，释放已经持有的资源
- 示例：从DVD读文件入磁盘，然后打印
  - 所需资源包括DVD驱动器、磁盘文件、打印机
  - 法1：一次请求三个资源
  - 法2：先请求DVD和磁盘，将数据读入磁盘；然后释放DVD和磁盘，再请求磁盘和打印机，执行打印
- 缺点：资源利用率低；有可能出现饥饿。

预防死锁：破坏“不可抢占”条件

- 如果进程处于阻塞状态，则所有它所持有的资源可以被抢占
- 这意味着：进程在请求资源时，
  - 如果请求的资源被处于阻塞状态的进程占用，则可以抢占，分配给请求进程
  - 如果请求的资源被处于非阻塞状态的进程占用，则不可以抢占（why？）
- 常用的应用范围：内存、CPU寄存器，不能应用于信号量

预防死锁：破坏“循环等待”条件

- 资源编号，定义“优先级”
- 约定每个进程必须先申请优先级低的资源，再申请优先级高的资源
- 如果要申请同一优先级的多个资源，必须一次性全部申请
- 优先级一般反映资源的使用次序

目录

为什么提出避免策略?

- 预防策略限制了资源利用率
  - 过于保守
- 避免策略添加的限制较弱
  - 要求进程预先声明需要资源的最大数量
  - 将系统划分为“安全状态”和“不安全状态”
  - 若某申请会导致系统进入不安全状态，则拒绝
  - 预防策略与避免策略类似于红绿灯与交警的区别

安全状态

- 已知每个进程请求的资源的数量的最大值
- 一个状态称为安全状态当且仅当存在一个安全序列$P_1, P_2, \cdots, P_n$使得
  - 对于任意进程P_i，P_i允许申请的最大资源数不超过系统剩余的资源加上所有P_j持有的资源（$j<i$）
- 注意
  - 只要找到一个这样的序列即可
  - 如果找不到此序列，则为不安全状态

安全状态、不安全状态、死锁

示例1：单资源

- 三个进程，12M内存，假如某个时刻的分配状态如下（还剩余3M内存），此状态是安全状态吗？

进程	最大需求	已分配
P_1	10	5
P_2	4	2
P_3	9	2

- 此时假如
  - P_1申请1M内存，可以分配给它吗？
  - P_2申请1M内存，可以分配给它吗？

示例2：三类资源的情况

- 假如A、B、C 资源总数分别为10, 5, 7，当前状态是安全的吗?

进程	最大需求	已分配
P_1	7 5 3	0 1 0
P_2	3 2 2	2 0 0
P_3	9 0 2	3 0 2

讨论：算法设计问题

- 给定资源分配情况和资源剩余情况，判断是否处于安全状态

进程	最大需求	已分配
P_1	7 5 3	0 1 0
P_2	3 2 2	2 0 0
P_3	9 0 2	3 0 2

进程	最大需求	已分配
P_1	10	5
P_2	4	2
P_3	9	2

- 要这种算法有何用？
  - 如果一个资源申请会导致系统进入不安全状态，则拒绝此申请；否则满足此申请

银行家算法

核心：安全性检测算法

- 数据结构：
  - 矩阵N：行为进程、列为资源、元素为最大剩余需求
  - 向量A：剩余资源
- 算法：
  - 在N中找一行，每个元素均≤A，如果没有，不安全
  - 否则，假设进程已执行完，从N中删除，归还它的资源到A中
  - 重复上述步骤，直到所有进程被删除，此时为安全
- 联想：像不像拓扑排序？

银行家算法

- 对任意请求，首先判断是否合法
  - 即，是否超过原先的最大申请量
- 其次，如果分配给它，利用安全性检测算法判断系统安全否?
  - 安全，则分配;
  - 否则，不予分配

单一资源的情况

- 三个进程，12M内存，假如某个时刻的分配状态如下（还有3M剩余内存），此状态安全吗？

进程	最大需求	已分配
P_1	10	5
P_2	4	2
P_3	9	2

  - 可以满足P_2，然后满足P_1，最后满足P_3，所以是安全的

示例：三类资源的情况

- 假如A、B、C 资源总数分别为10, 5, 7，当前状态是安全的吗?

进程	最大需求	已分配
P_0	7 5 3	0 1 0
P_1	3 2 2	2 0 0
P_2	9 0 2	3 0 2
P_3	2 2 2	2 1 1
P_4	4 3 3	0 0 2

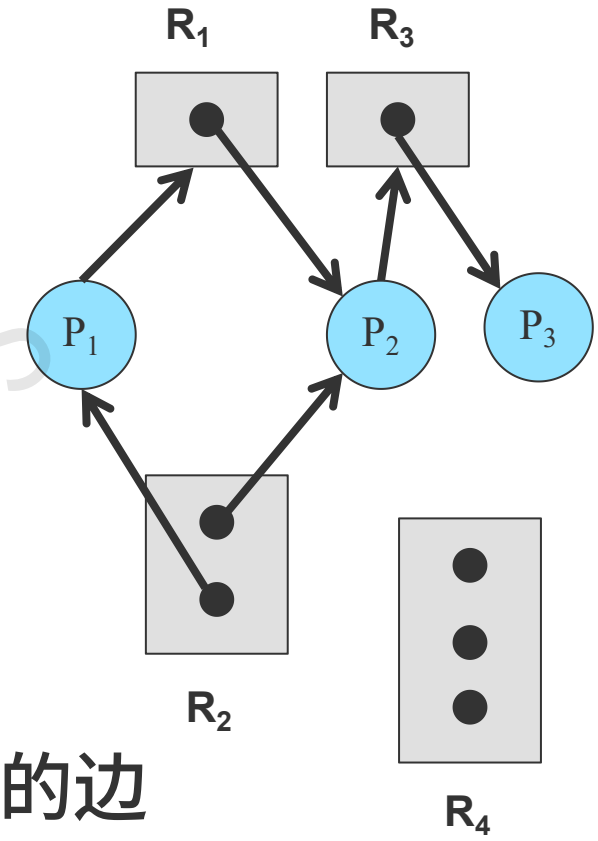
死锁避免策略评价

- 比预防策略效率高
- 但是，实用价值有限
  - “真的不知道自己需要多少资源！”

目录

如何检测死锁?

- 资源分配图
  - 资源：矩形，黑点表示资源
  - 进程：圆形
  - 请求：从圆到矩形画边
  - 分配：从矩形到圆画边



- 资源分配图的化简
  - 重复：找不阻塞的进程，删它的边
- 死锁定理：死锁的充分条件是，资源分配图不可完全简化。

死锁的检测算法

- **数据结构：**
  - 矩阵R：行为进程、列为资源，元素为需求
  - 向量A：剩余资源
- **算法：**
  - 在R中找一行，每个元素均≤A，如果没有，死锁
  - 否则，假设它已执行完，从R中删除，归还它的资源到A中
  - 重复上述步骤，直到所有进程被删除，此时无死锁

- **和银行家算法中的安全状态检测有何区别？**

关于检测算法的两个问题

- 什么时候启动检测算法?
  - 每次申请资源时启动?
  - 周期性启动?
  - 当CPU使用率低于40%时启动?
- 死锁检测算法与银行家算法中“安全状态检测”有何联系和区别?
  - 安全状态检测针对了“最坏”请求，非实际请求
  - 死锁检测针对实际请求

死锁的解除

- 剥夺资源
  - 从其它进程剥夺足够数量的资源给死锁的进程，以解除死锁
- 撤消进程
  - 撤销全部死锁进程
  - 或者按某顺序逐个撤消进程，直至有足够的资源可用、死锁状态消除为止
  - 有一些优化策略
    - 如，为解除死锁状态所需要撤消的进程数目最小；或者撤消进程所付出的代价最小

死锁总结

- 死锁的必要条件
  - 互斥、不可抢占、请求和保持、循环等待
- 死锁的应对策略
  - 预防：破坏必要条件
  - 避免：防止进入不安全状态（银行家算法）
  - 检测与解除：允许进入死锁，然后检测并解除
  - 忽略策略：do nothing

本部分内容的要求（2025考研大纲）

- 死锁的基本概念
- 死锁预防
- 死锁避免
- 死锁检测和解除

安全性检测算法具体描述：算法的输入

- 可利用资源向量Available
  - 含有m个元素的数组，Available[j]＝k表示系统中现有R_j类资源k个。
- 最大需求矩阵Max
  - 是一个n*m的矩阵，Max(i，j)＝k表示进程i需要R_j类资源的最大数目为k
- 分配矩阵Allocation
  - 这是一个n*m的矩阵，如果Allocation(i，j)＝k表示进程i当前已分得R_j类资源的数目为k。
- 需求矩阵Need
  - 即为 Max-Allocation

安全性检测算法具体描述：算法步骤

1. 设置两个向量
   - Work：系统当前资源数目。Work=Available。
   - Finish：进程是否顺利运行结束。Finish[i] =fa1se。
2. 寻找到一个能满足下述条件的进程i：
   - Finish[i] = fa1se；Need[i,j] <= Work[j]；
3. 如找到，则
   - Finish[i]=true; Work[j]+ =Allocation[i,j];
   - 继续执行2.
4. 否则,若所有进程均已结束，则安全；否则不安全