

Lecture 5: 知识 - 死锁

死锁的概念

死锁的概念

概念：死锁基本概念

- 概念：死锁
 - 在并发环境下，各进程因**竞争资源**而造成的一种互相等待对方手里的资源，导致**各进程都阻塞**，都无法向前推进的现象；
- 概念：三类基本概念
 - 死锁：各进程**互相**等待对方手里的资源，导致各进程都阻塞，无法向前推进的现象；
 - 饥饿：由于长期得不到想要的资源，**某进程**无法向前推进的现象；
 - 死循环：某个进程执行过程中一直跳不出某个循环的现象。有时是因为程序逻辑 bug 导致的，有时是程序员故意设计的；
 - 共同点：进程发生异常状态，无法继续进行下去；
- 区别：死锁、饥饿、死循环
 - 关于死锁：
 - 死锁进程 -> 一定处于 **阻塞态**；
 - 产生死锁时 -> 一定至少有两个及以上进程同时发生死锁；
 - 关于饥饿：
 - 可能只有一个进程发生饥饿；
 - 发生饥饿的进程可能长期为 **阻塞态** 或者 **就绪态**；
 - 关于死循环：
 - 死循环进程 -> 可以在 **运行态** 上；
 - 死循环的产生 -> 由代码逻辑错误导致；
- 补充：活锁
 - 资源未被占用，进程双方互相礼让，导致永远等待下去；
- 图示：

	共同点	区别
死锁	都是进程无法顺利向前推进的现象（故意设计的死循环除外）	死锁一定是“循环等待对方手里的资源”导致的，因此如果有死锁现象，那 至少有两个或两个以上的进程同时发生死锁 。另外，发生死锁的进程一定处于阻塞态。
饥饿		可能只有一个进程发生饥饿 。发生饥饿的进程既可能是阻塞态(如长期得不到需要的I/O设备)，也可能是就绪态(长期得不到处理机)
死循环		可能只有一个进程发生死循环。死循环的进程可以上处理机运行（可以是运行态），只不过无法像期待的那样顺利推进。死锁和饥饿问题是由于操作系统分配资源的策略不合理导致的，而死循环是由代码逻辑的错误导致的。 死锁和饥饿是管理者（操作系统）的问题，死循环是被管理者的问题。

死锁产生条件

概念：死锁产生的必要条件

- 产生死锁必须同时满足以下 4 个条件：
- 1. 互斥条件：
 - 只有对必须互斥使用的资源的争抢才会导致死锁；
 - 像内存，扬声器这样可以同时让多个进程使用的资源是不会导致死锁的（因为进程不用阻塞等待这种资源）；
- 2. 不剥夺条件：
 - 进程所获得的资源在未使用完之前，不能由其他进程强行夺走，只能主动释放；
- 3. 请求和保持条件：
 - 进程已经保持了至少一个资源，但又提出了新的资源请求，而资源又被其他进程占有，此时请求进程被阻塞，但又对自己已有的资源保持不放；
- 4. 循环等待条件：
 - 存在一种进程资源的循环等待链，链中的每一个进程已获得的资源同时被下一个进程所请求。
- 记忆：
 - 我的东西（互斥）不能被别人抢走（不剥夺），并且还要更多（请求和保持），得不到就一直等（循环等待）
- 注意：
 - 发生死锁时一定有循环等待，但是发生循环等待时未必死锁（循环等待是死锁的必要不充分条件）
 - 例如：同类资源数大于 1，即使有循环等待，也未必发生死锁

死锁处理策略

概念：死锁的处理策略

- 概念：三种策略
 - 1. 预防死锁
 - 破坏死锁产生的 4 个必要条件中的一个或几个，不会给可能导致死锁的进程分配资源；
 - 死锁预防通过破坏死锁必要条件，防止死锁的发生；
 - 举例：顺序资源分配法 -> 限制资源顺序
 - 2. 避免死锁
 - 用某种方法防止系统进入不安全状态，从而避免死锁，会给可能导致死锁的进程分配资源；
 - 银行家算法属于死锁避免的方法，可以判断当前是否处于安全状态，但不能判断系统是否处于死锁状态；
 - 举例：银行家算法
 - 3. 死锁的检测和解除
 - 允许死锁的发生，不过操作系统会负责检测出死锁的发生，然后采取某种措施解除死锁；
 - 不需要进程运行所需的资源信息；
 - 举例：资源分配图 + 进程撤销/资源剥夺

• 分析：真题 - 2015年

• 题目：

【2015 统考真题】若系统 S_1 采用死锁避免方法， S_2 采用死锁检测方法。下列叙述中，正确的是（ ）。

- I. S_1 会限制用户申请资源的顺序，而 S_2 不会
- II. S_1 需要进程运行所需的资源总量信息，而 S_2 不需要
- III. S_1 不会给可能导致死锁的进程分配资源，而 S_2 会

- A. 仅 I、II B. 仅 II、III C. 仅 I、III D. I、II、III

• 解析：

- I：顺序资源分配法（属于死锁预防）通过限制用户申请资源的顺序来实现死锁预防，而不是死锁避免或死锁检测，I 错误；
- II：银行家算法属于死锁避免方法，需要进程运行所需资源总量信息，II 正确；
- III：死锁避免方法不会给可能导致死锁的进程分配资源，而死锁检测方法会给可能导致死锁的进程分配资源，III 正确，故本题选 B；

• 分析：真题 - 2019年

• 题目：

【2019 统考真题】下列关于死锁的叙述中，正确的是（ ）。

- I. 可以通过剥夺进程资源解除死锁
- II. 死锁的预防方法能确保系统不发生死锁
- III. 银行家算法可以判断系统是否处于死锁状态
- IV. 当系统出现死锁时，必然有两个或两个以上的进程处于阻塞态

- A. 仅 II、III B. 仅 I、II、IV C. 仅 I、II、III D. 仅 I、III、IV

• 解析：

- I：剥夺进程资源属于死锁解除的方法，I 正确；
- II：死锁预防通过破坏死锁必要条件，防止死锁的发生，II 正确；
- III：银行家算法属于死锁避免的方法，可以判断当前是否处于安全状态，但不能判断系统是否处于死锁状态，III 错误；
- IV：单个进程处于阻塞态无法形成死锁，不可能满足循环等待条件，因此当系统出现死锁时。因此产生死锁时，必然有**两个或两个以上的进程**处于阻塞态，IV 正确；

预防死锁

破坏互斥条件

破坏条件

• 互斥条件

- 只有对必须互斥使用的资源的争抢才会导致死锁；

• 解除方法：

- 如果把只能互斥使用的资源改造为允许共享使用，则系统不会进入死锁状态；
- 比如：SPooling技术
- 操作系统可以采用 SPooling 技术把独占设备在逻辑上改造成共享设备；
- 比如，用 SPooling 技术将打印机改造为共享设备...

• 策略缺点：

- 1. 并不是所有的资源都可以改造成可共享使用的资源；
- 2. 为了系统安全，很多地方还必须保护这种互斥性；
- 结论：
 - 很多时候都无法破坏互斥条件；
- 图示：



破坏不剥夺条件

破坏不剥夺条件

- 不剥夺条件：
 - 进程所获得的资源在未使用完之前，不能由其他进程强行夺走，只能主动释放；
- 方案一
 - 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；
 - 结果 -> 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
 - 当某个进程需要的资源被其他进程所占有时，可以由操作系统协助，将想要的资源强行剥夺；
 - 这种方式一般需要**考虑各进程的优先级**（比如：剥夺调度方式，就是将处理机资源强行剥夺给优先级更高的进程使用）
- 缺点：
 - 1. 实现起来比较复杂；
 - 2. 释放已获得的资源可能造成**前一阶段工作的失效**；
 - 因此这种方法一般只适用于易保存和恢复状态的资源，如 CPU
 - 3. 反复地申请和释放资源会增加系统开销，**降低系统吞吐量**；
 - 4. 若采用方案一，意味着只要暂时得不到某个资源，之前获得的那些资源就都需要放弃，以后再重新申请。如果一直发生这样的情况，就会**导致进程饥饿**；

破坏请求和保持条件

概念：静态分配方法

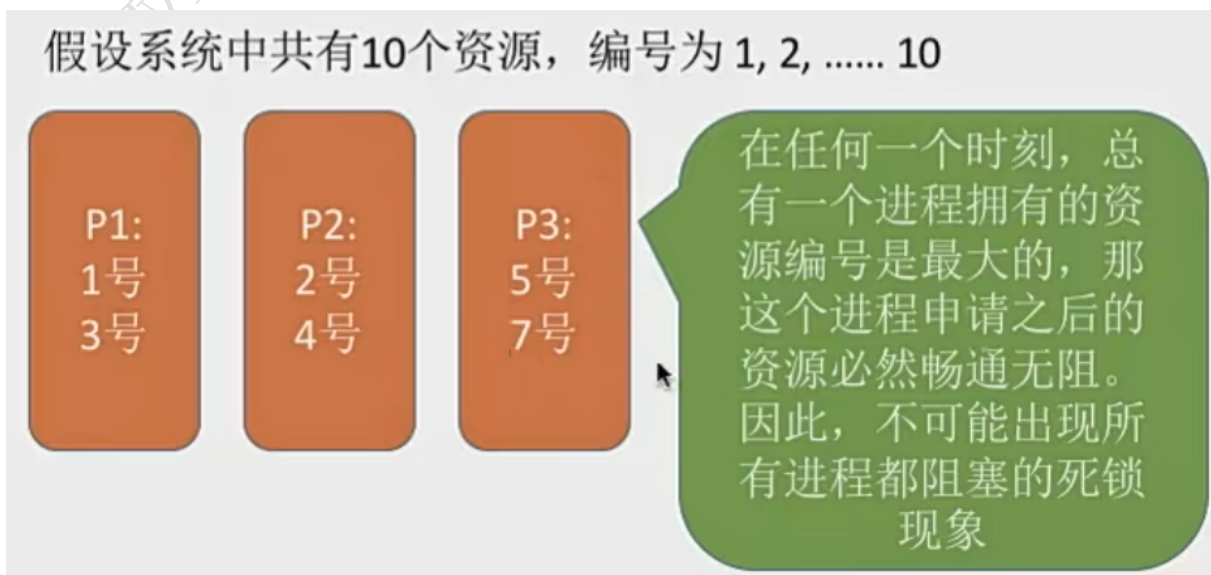
- 概念：
 - 请求和保持条件：进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源又被其他进程占有，此时请求进程被阻塞，但又对自己已有的资源保持不放；

- 方案：
 - 可以采用**静态分配方法**；
 - 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，不让他投入运行；
 - 一旦投入运行后，这些资源就一直归它所有，该进程就不会再请求别的任何资源；
- 特点：
 - 优点：
 - 实现简单
 - 缺点：
 1. 有些资源可能只需要用很短的时间，因此如果进程的整个运行期间都一直保持着所有资源，就会**造成严重的资源浪费**，资源利用率极低；
 2. 该策略也有可能**导致某些进程饥饿**；

破坏循环等待条件

概念：顺序资源分配法

- 循环等待条件：
 - 存在一种进程资源的循环等待链，链中的每一个进程已获得的资源同时被下一个进程所请求；
- 方案：
 - 可以使用顺序资源分配法；
 - 首先给系统中的资源编号，规定每个进程必须**按编号递增的顺序请求资源**，同类资源 (即编号相同的资源)一次申请完；
- 分析：
 - 一个进程只有已占有小编号的资源时，才有资格申请更大编号的资源；
 - 按此规则，已持有大编号资源的进程不可能逆向地回来申请小编号的资源，从而就不会产生循环等待的现象；
- 图示
 - 在任何一个时刻，总有一个**进程拥有的资源编号是最大的**，那这个进程申请之后的资源必然畅通无阻因此，不可能出现所有进程都阻塞的死锁现象；



缺点

- 1. 不方便增加新的设备，因为可能需要重新分配所有的编号；
- 2. 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；
- 3. 必须按规定次序申请资源，用户编程麻烦；

避免死锁

概念

安全序列

- 如果系统按照这种序列分配资源，则每个进程都顺利完成；
- 只要能找出一个安全序列 $\rightarrow$ 系统就是安全状态；
- 注意：安全序列可能有多个；

不安全状态

- 如果分配资源之后，系统中找不出任何一个安全序列，系统就进入了不安全状态；
- 不安全状态 $\rightarrow$ 之后可能所有进程都无法顺利的执行下去；
- 注意：如果有进程提前归还了一些资源，那系统也有可能重新回到安全状态，不过在分配资源之前总是要考虑到最坏的情况；

与死锁的联系

- 如果系统处于安全状态，就一定不会发生死锁。
- 如果系统进入不安全状态，就可能发生死锁(处于不安全状态未必就是发生了死锁，但发生死锁时一定是在不安全状态)

银行家算法

概念：核心思想

- 概念：
 - 在进程提出资源申请时，先**预判**此次分配是否会导致系统进入不安全状态。如果会进入不安全状态，就暂时不答应这次请求，让该进程先阻塞等待；
- 注意：
 - **真题**：2019年
 - 银行家算法属于死锁避免的方法，可以判断当前是否处于安全状态，但不能判断系统是否处于死锁状态

分析举例

- 尝试找出一个安全序列.. 依次检查剩余可用资源 (3,3,2)是否能满足各进程的需求
- 计算方法：

进程	最大需求	已分配	最多还需要
P0	(7, 5, 3)	(0, 1, 0)	(7, 4, 3)
P1	(3, 2, 2)	(2, 0, 0)	(1, 2, 2)
P2	(9, 0, 2)	(3, 0, 2)	(6, 0, 0)
P3	(2, 2, 2)	(2, 1, 1)	(0, 1, 1)
P4	(4, 3, 3)	(0, 0, 2)	(4, 3, 1)

• 需求图：

进程	最大需求	已分配	最多还需要
P0	(7, 5, 3)	(0, 1, 0)	(7, 4, 3)
P1	(3, 2, 2)	(2, 0, 0)	(1, 2, 2)
P2	(9, 0, 2)	(3, 0, 2)	(6, 0, 0)
P3	(2, 2, 2)	(2, 1, 1)	(0, 1, 1)
P4	(4, 3, 3)	(0, 0, 2)	(4, 3, 1)

• 分析过程：

- 此时系统是否处于安全状态？
- 思路：
 - 尝试找出一个安全序列..{P 1, P 3, P0, P 2, P 4}
 - 依次检查剩余可用资源 (3,3,2)是否能满足各进程的需求可满足 P 1 需求，将 P 1 加入安全序列，并更新剩余可用资源值为 (5,3,2)
 - 依次检查剩余可用资源 (5,3,2)是否能满足剩余进程 (不包括已加入安全序列的进程)的需求可满足 P 3 需求，将 P 3 加入安全序列，并更新剩余可用资源值为 (7,4,3)
 - 依次检查剩余可用资源 (7,4,3)是否能满足剩余进程 (不包括已加入安全序列的进程)的需求..
- 以此类推，共五次循环检查即可将 5 个进程都加入安全序列中，最终可得一个安全序列。该算法称为安全性算法。可以很方便地用代码实现以上流程，每一轮检查都从编号较小的进程开始检查

• 手算思路：

- 经对比发现，(3,3,2) 可满足 P 1、P 3，说明无论如何，这两个进程的资源需求一定是可以依次被满足的；
- 因此 P 1、P 3 一定可以顺利的执行完，并归还资源；
- 此时可把 P 1、P 3 先加入安全序列 $\rightarrow (2, 0, 0) + (2, 1, 1) + (3, 3, 2) = (7, 4, 3)$
- 此时分配完后的状态：

进程	最大需求	已分配	最多还需要
P0	(7, 5, 3)	(0, 1, 0)	(7, 4, 3)
P2	(9, 0, 2)	(3, 0, 2)	(6, 0, 0)
P4	(4, 3, 3)	(0, 0, 2)	(4, 3, 1)

- 接下来剩下的 P 0、P 2、P 4 都可被满足。同理，这些进程都可以加入安全序列。于是，5 个进程全部加入安全序列，说明此时系统处于安全状态，暂不可能发生死锁

数据结构

- **Available**: 长度为 m 的一维数组 Available 表示还有多少可用资源
- **Max**: $n \times m$ 矩阵 Max 表示各进程对资源的最大需求数
- **Allocation**: $n \times m$ 矩阵 Allocation 表示已近给各进程分配了多少资源
- **Need**: $\text{Max} - \text{Allocation} = \text{Need}$ 矩阵表示各进程最多还需要多少资源
- **Request**: 用长度为 m 的一维数组 Request 表示进程此次申请的各种资源数
- 图示:



银行家算法步骤

- 图示:

可用**银行家算法**预判本次分配是否会导致系统进入不安全状态：

因为它所需要的资源数已超过它所宣布的最大值。

①如果 $Request_{ij} \leq Need_{ij}$ ($0 \leq j \leq m$) 便转向②；否则认为出错。

②如果 $Request_{ij} \leq Available_{ij}$ ($0 \leq j \leq m$)，便转向③；否则表示尚无足够资源， P_i 必须等待。

③系统**试探着**把资源分配给进程 P_i ，并修改相应的数据（并非真的分配，修改数值只是为了做预判）

$Available = Available - Request_i$;

$Allocation[i, j] = Allocation[i, j] + Request_{ij}$;

- $Need[i, j] = Need[i, j] - Request_{ij}$

④操作系统执行**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，才正式分配；否则，恢复相应数据，让进程阻塞等待。

1. 检查此次申请是否超过了之前声明的最大需求数
2. 检查此时系统剩余的可用资源是否还能满足这次请求
3. 试探着分配，更改各数据结构
4. 用安全性算法检查此次分配是否会导致系统进入不安全状态

安全性算法步骤

- 检查当前的剩余可用资源是否能满足某个进程的最大需求，如果可以，就把该进程加入安全序列，并把该进程持有的资源全部回收。
- 不断重复上述过程，看最终是否能让所有进程都加入安全序列。

注意

- 系统处于不安全状态未必死锁，但死锁时一定处于不安全状态。
- 系统处于安全状态一定不会死锁。

检测和解除

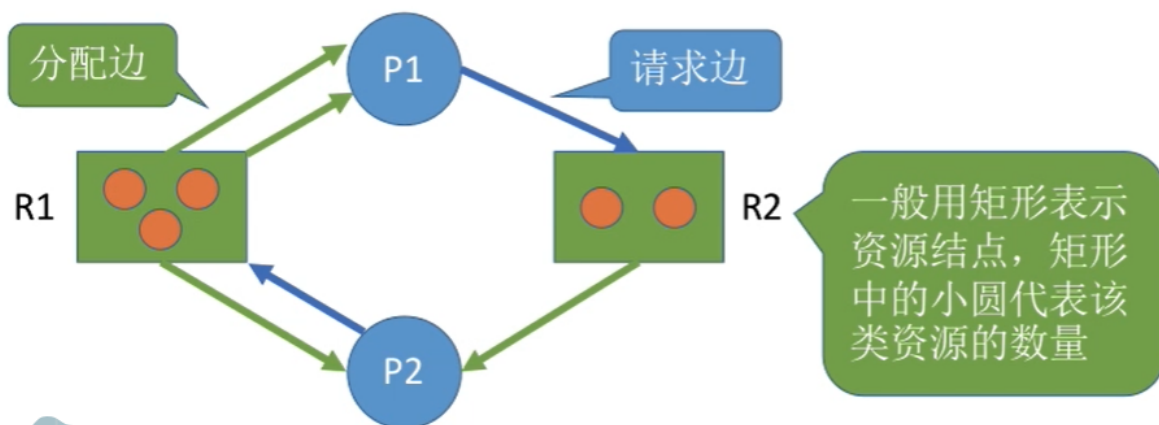
死锁的检测

引入：死锁的检测

- 概念：
 - 为了能对系统是否已发生了死锁进行检测，必须：
 1. 用某种数据结构来**保证资源的请求和分配信息**；
 2. 提供一种算法，利用上述信息来**检测系统是否已进入死锁状态**；
- 核心：
 - **依次消除与不阻塞进程相连的边，直到无边可消**
- 内容
 - 两种结点
 - 进程结点：对应一个进程；
 - 资源结点：对应一类资源，一类资源可能有多个；
 - 两种边
 - 进程结点 $\rightarrow$ 资源结点：请求边
 - 表示进程想申请几个资源（每条边代表一个）
 - 资源结点 $\rightarrow$ 进程结点：分配边

- 表示已经为进程分配了几个资源（每条边代表一个）

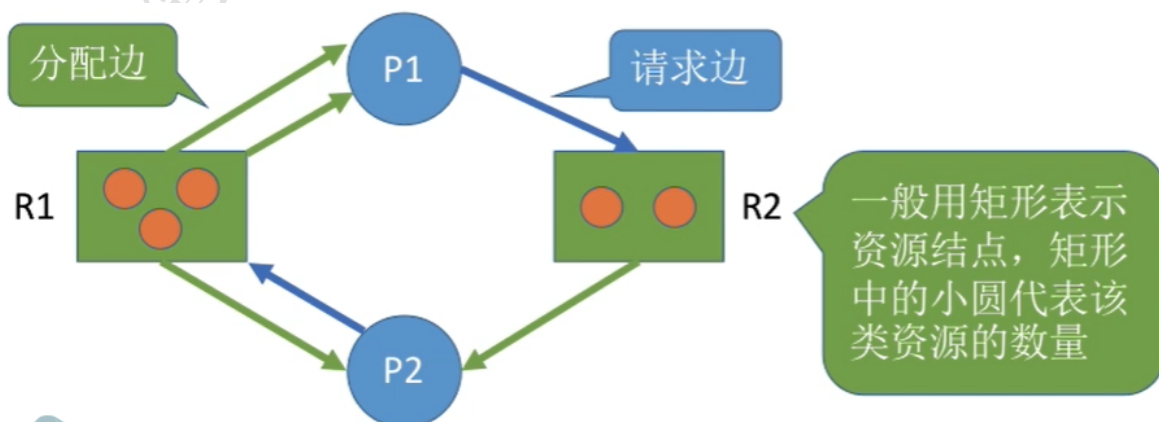
• 图示：



检测死锁的算法

- 1. 在资源分配图中，找出既不阻塞又不是孤点的进程 P_i
 - 即找出一条有向边与它相连，且该有向边对应资源的申请数量小于等于系统中已有空闲资源数量。
 - 如下图中，R1 没有空闲资源，R2 有一个空闲资源。
 - 若所有的连接该进程的边均满足上述条件，则这个进程能继续运行直至完成，然后释放它所持有的所有资源。
 - 消除它所有的请求边和分配边，使之成为孤立的结点
 - 在下图中，P1 是满足着一条件的进程结点，于是将 P1 的所有边消去。
- 2. 进程 P_i 所释放的资源，可以唤醒某些因等待这些资源而阻塞的进程，原来的阻塞进程可能变为非阻塞进程。
 - 下图中，P2 就满足这样的条件
 - 根据 (1) 中的方法进行一系列简化后，若能消去图中所有的边，则称该图是可完全简化的；
 - 能够按照 $P1 \rightarrow P2$ 的顺序执行结束；

• 图示：



- 注意：死锁检测算法概括：依次消除与不阻塞进程相连的边，直到无边可消

死锁的解除

本质：并不是系统中所有的进程都是死锁状态，用死锁检测算法化简资源分配图后，还连着边的那些进程就是死锁进程

解除死锁的主要方法：

- 1. 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；
- 2. 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，一旦被终止则功亏一篑；
- 3. 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**；

解除死锁方法选择原则：

1. 进程优先级
2. 已执行多长时间
3. 还要多久能完成
4. 进程已经使用了多少资源
5. 进程是交互式的还是批处理式的