并发Bug&死锁

Courses	操作系统
Done	☐
Status	Done

死锁

【例如】

1. AA-Deadlock：在持有一把锁的时候再次获得这把锁
2. **ABBA-Deadlock（最常见的）**：$T_1 - lock(A) - lock(B)$

   $T_2 - lock(B) - lock(A)$

   哲学家吃饭问题

死锁产生的必要条件

1. 得到球才能继续：互斥条件
2. 得到球的人还想要更多的球：持有并等待条件
3. 不能抢别人的球：不剥夺条件
4. 循环等待：循环等待条件

4是我们优先考虑实现的

如何实现?

- lock ordering，所有程序按照从小到大的顺序获得锁

数据竞争—Data race

定义：不同的线程同时访问同一内存，且至少一个是写

谁限制性会对系统的状态产生**非确定性**的影响

死锁检测算法—RAG

资源分配图

- 节点：
  - 进程节点
  - 资源节点
- 边：
  - 请求边：进程节点→资源
  - 分配边：资源节点→进程

如果进程-资源分配图中无环路，则此时系统没有发生死锁

有环路+每个资源类中仅有一个资源—有死锁

死锁检测算法的步骤

进程数：n，资源种类数：m

- available向量：长度为m，表示目前每种资源可供分配的数目
- allocation矩阵：当前已分配资源的情况
- max最大需求矩阵
- 当前需求矩阵Need（也叫资源申请矩阵Request）：Need=Max-allocation
- 辅助向量finish & work

过程：

- 初始化work=available， finish[*]=false
- 找到一个进程k，使得need[k,*]≤work[*]
  - 如果存在，则将finish[k]标记为true，work[*]=work[*]+allocation[k, *]
    - 继续寻找k
  - 否则，直接前往最后一步判断
- 如果还有finish标记为false，则存在死锁

死锁避免

银行家算法

- 资源总量矩阵Resource
- 当前可用数量矩阵Available
- 最大需求矩阵Claim
- 已分配矩阵Allocation
- 尚需资源矩阵Need = Claim-allocation
- 当前申请矩阵Request

相关描述

- 如果要启动一个新进程，则保证所有进程的Claim之和不大于Resource
- 进程序列是安全的：尚需 ≤ 当前可用+已分配（need ≤ available + allocation）

算法的思想

首先尝试分配给进程k，request[k]

转向安全测试算法↑（RAG），如果通过，则安全，否则不安全，撤回分配