08 分布式互斥

- 互斥锁需求
  - 正确性：最多只有一个进程获得锁，进入临界区
  - 公平性：进程不能一直获得锁，超时要释放
    - 最终公平：不会一直被排斥
    - 界限公平：在一定周期内，一定会获得锁
- 分布式需求
  - 低的消息损耗
  - 没有瓶颈
  - 允许乱序消息
    - 今天只关注前三个条件
  - Allow processes to join protocol or to drop out
  - 容忍进程失败
  - 容忍消息丢失
- 中心化的互斥
  - 所有客户都向中心申请锁
  - 中心把锁分配给当前的申请者，或者等待队列中的进程
- Bully选举
  - 进程发现leader失败后
  - 发送选举消息，包含一个(当前进程的一个id)，如果没人响应进程赢得选举
  - 如果收到更大id的响应，P结束leader
  - 如果收到小id的选举请求，P发送结束，P重新开始选举
    - **可能会进行多次选举啊，甚至同时进行多次选举**
    - **小id的进程收到大id的选举请求没有回复，怎么确定不是大进程自己断网了**
- 去中心化算法
  - 获得m(m>n/2)个节点的同意
  - 每个节点收到请求后，立刻回复同意或拒绝
  - 节点失败的问题：节点重启后可能忘记投票结果
  - 如果得到的票少于m
    - 之后再试
    - 如果大家都请求影响性能
    - 饥饿
- Lamport Totally-Ordered Multicasting
  - 基于lamport时钟，将事件id按进程划分为n个区间(取余)
    - **可以将事件id，划分为两个区间，高位区间事件id自增，低位用于进程id**
    - 可以预留，以及容忍失败
    - 每个进程的事件编号只能在当前进程的区间增加

        - 进程间通信之后，进程取更大的一个事件id作为基本
        - 假设所有消息都按照发送顺序到达，且不丢消息
    - 每个进程需要独占时都向其他所有进程发送请求
    - 其他进程收到请求后，把ack消息发送给所有其他进程
        - 每个进程都知道当前谁在独占，谁在等待
    - 等待队列按事件id排序
        - 如果有进程不按规则行事容易产生饥饿
- Lamport Mutual Exclusion
    - 基于Lamport Totally-Ordered Multicasting
    - 只向请求者ack，
    - 独占结束后，再release
    - 所有进程都知道请求队列有谁
        - 但是不知道队头到底在不在临界区
    - 每次独占需要3*(n-1)次消息传递
        - 第一次向其他进程请求
        - 第二次接收其他进程ack
        - 第三次释放
- Ricart & Agrawala Mutex
    - 基于Lamport Mutual Exclusion
    - 当期进程如果也要请求独占，且当期进程的请求编号小，就暂时不发送ack
    - 否则，直接回复OK
    - 拿到其他所有进程的确认后，可以直接进入临界区
    - 请求独占需要2*(n-1)次消息
        - 请求+其他进程确认
        - 不用释放了，因为下次请求来的时候，回复ack，就表明释放了
    - 错误容忍的问题
        - 如果当期在临界区失败了，且队列有其他进程请求
        - 可能系统再也无法进入独占
- Token Ring Mutual Exclusion
    - 令牌在进程间传递
    - 令牌在手就可以进入独占，否则等待
    - 性能不可控
    - 错误容忍
        - 令牌机器失败，丢失令牌
-