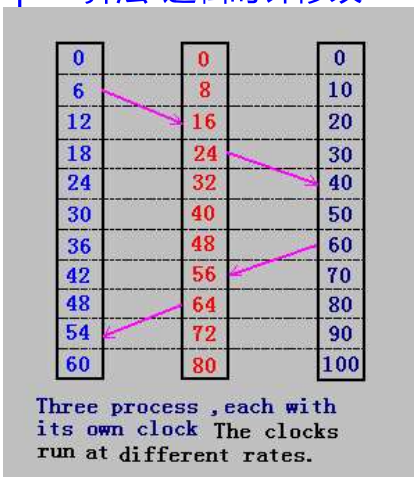
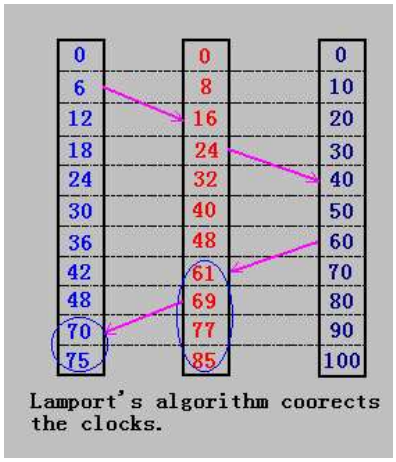
- 三、分布式同步算法
  - 逻辑时钟(Logical Clocks)
    - 时间在同步中起重要的作用，首先，我们来看分布式系统中时间的度量(measured)。
    - 几乎所有的计算机是使用电路来记录时间，尽管普遍使用时钟这个概念来表示这些设备，但它们并不是真正意义是的时钟(clock)，更确切地说，计算机中使用的只是计时器(timer)，由有规律振荡的晶体来产生。因为加工工艺等原因，不可能有两个晶体时钟是一致，从而不同的计算机其时钟不可能完全一样。
    - 假使两台或几台机器时钟可以非常精确，但大型分布式系统允许两台机器仅次于地球的两个时区上，这样它们的时钟仍是不一致的。
    - 因此，在分布式系统中，需要考虑另一种时间的度量，即逻辑时钟。在这方面有重大贡献的是Lamport，他在1978年提出时钟同步是可能的，并给出一些算法，这些观点为分布式系统奠定了重要的理论基础。
  - 逻辑时钟(logical clock)也称为Lamport 算法
    - Lamport指出：时钟同步不需要绝对的(精确时间)，如果进程间没有相互作用，就不需要时钟同步。并且进程所关心的不是什么时间做什么工作(由于进程的运行过程由多种因素决定，有不确定性)，它们所关心的是事件发生的顺序。
    - 思想
      - **happen before:**
        - 如果事件a是在事件b之前发生，则记a->b,即a happen before b。特别地，在一次通信中，如果发送进程发送消息的事件是a，接收进程接收消息的事件为b，则a->b。
        - happen before是可传递的，即如果a->b,b->c则a->c。
      - timestamp：
        - 在分布式系统中，每一台机器都设置一个整型变量作为“时钟”(它并不是真实的时钟)，当一个事件a发生时，用这个时钟的值作为事件a的timestamp(时间戳),记作C(a)。
        - 这样，如果a->b,则C(a)<C(b);
        - 在同一机器上的任两个事件a和b，都有C(a)!=C(b);
        - 在进程通信中，消息中含有发送时的时间戳，消息到达目标机器时，它的内核检查其时间戳，如果它的当前时钟小于所收到消息的时间戳，则修改它的时钟，使其时钟值大于接收消息的时间戳。
      - Lamport算法-逻辑时钟修改
      - 



- 四、分布式互斥算法
  - 集中式算法
    - 在分布式系统中最直接的方法是模仿单处理器系统中的作法实现五斥。
    - 基本思想
      - 指定一个进程为协调者(Coordinator),当一个进程要进入临界区时，它发送一个请求消息给协调者进程表示它的要进入临界区并要求给一个许可(permission)，如果当前没有进程在临界区执行，协调者进程返回一介许可的应答，当许可的应答收到时，要求进程则可以进入执行，当它的临界区执行完成时，再发送一个要求释放临界区的消息。
    - 特点
      - 可以保证临界区的互斥;
      - 具有分平性(fare);
      - 容易实现;
      - 可以用于其他的资源分配;
      - 协调者进程的错误将使用整个系统无法工作;
      - 死锁不容易检测
      - 协调者进程可能出现瓶颈。
  - Ricart & Agrawala's算法
    - 基本思想
      - 当一个进程要进入临界区时，它建立一个包含它要进入的临界区名、进程号、和当前时间戳(timestamp)，并把消息发送给所有的进程，理论上也包括它自己，消息传送是可靠的，每个消息要求有确认，如果网络支持可靠的组通信，则可用于代替单个的通信。
      - 当一个进程收到另一个进程的一个请求时，它采取的工作依赖于消息中临界区名，这有三种情况:
        - 如果它不在相关临界区内，并且也不想进入，它返回一个许可(OK)的应答;
        - 如果它已经在临界区内，它不返回任何应答，而是将消息加入场专门的请求队列中;
        - 如果它想进入临界区执行(但前还没有进入)，它把消息中的时间戳，与它发送给其他进程的请求消息的时间戳进行比较，时间戳小的则获得进入临界区的许可。如果当前收到的消息的时间戳小，就发送一个OK消息，否则，如果进程自己发送的消息中的时间戳小，就不做发送任何消息而是将收到的请求加入请求队列中。
      - 进程在发送了请求进入临界区的请求消息后，就等待其他进程的许可(OK)消息，一旦它得到所有进程的许可(OK)，就可以进入临界区执行。
      - 进程退出临界区时，向在该进程的请求队列中的所有进程发送OK消息，并从队列中删除这些进程的请求消息。
    - 特点

        - 可以实现互斥，并保证不出现死锁(deadlock)或饿死(starvation);
        - 每次请求进入临界区要2(n-1)次通信(假定系统中的n个进程)。
        - 任何一个进程的崩溃，将使用算法无法完成。
    - 令牌环算法(Token Ring)
        - 基本思想
            - 将系统中所有进程建立一个逻辑环(logical ring)，环中进程的顺序可以按进程地址排序或其他方法排序，只要环中一进程能知道它的下一个进程是谁即可;
            - 环被初始化后，进程0得到一个令牌(Token)，令牌沿环逐个下传，从k到中k+1(当k+1为环中进程时取0);
            - 当一个进程从它的上一个得到令牌时， 如果它要进入临界区，则就可以进入执行，临界区执行完成后，再把令牌传给下一个。
            - 每获得一个令牌，至多只能执行一个临界区。
        - 特点
            - 可以实现互斥，不会出现饿死
            - 令牌丢失时要建立一个新的令牌，但很难确认是否已经丢失。
            - 进程崩溃。
    - 三种算法的比较(Comparison of the three Algorithms)
        -

	Message per entry/exit	Delay before entry(in message times)	problems
Centralized	**3**	**2**	**Coordinator crash**
Distributed	**2(n-1)**	**2(n-1)**	**Crash any process**
Token Ring	**1 to ∞**	**0 to n-1**	**Lost Token,process crash**

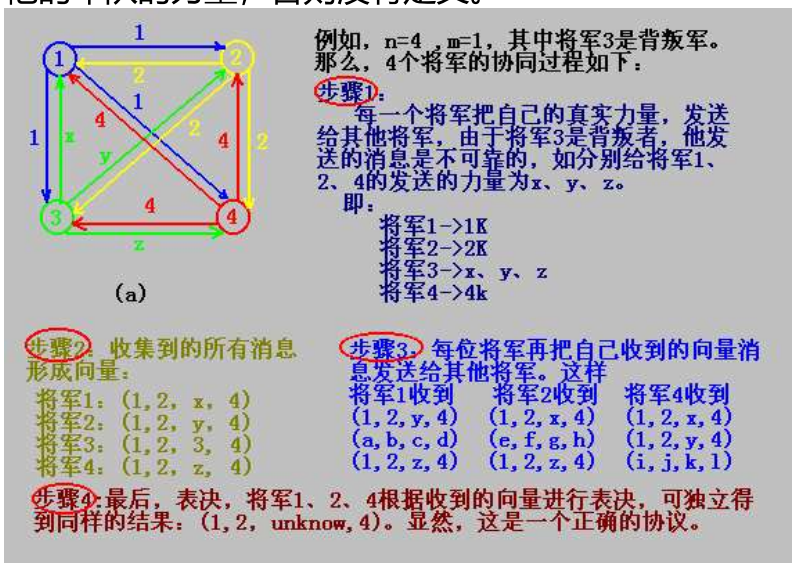
- 五、分布式系统的可靠性
    - 1.选择算法(Elect Algoriathms)
        - 在分布式系统的许多算法中都要有一个进程充当协调者、发起者、序列生成器、或其他特殊的角色，如集中式(互斥)算法中的协调者进程。通常，由哪个进程充当这个角色均可，但总得有一个进程来承担。这一节介绍选择这种协调者的算法即选择算法。
        - 几个假定
            - 系统中每一个进程都有一个唯一的标识，如进程号;
            - 系统中每个进程都知道其他进程的标识，但为些进程可能是活动的，也可能是关闭或崩溃;
            - 选择算法的目标：保证在选择算法执行后，所有进程都认可某个进程成为了新的协调者。
        - Bull算法
            - 当某个进程发现协调者(Coordinator)无法响应它的请求时，该进程就启动一个选择过程。进程P，执行的选择过程如下:
            - P向所有的进程号(优先级)比它高的进程发送Election消息;
            - 如果得不到其他进程的响应，进程P获胜，成为协调者;
            - 如果有一个进程号(优先级)更高的进程响应，该进程就接管选择过程。进程P的任务完成;

- 任何时刻进程都可能收到进程号(优先级)比它小的进程发送来的Election消息，该消息到达时，消息的接收者就向发送者发送一个OK消息，说明它仍是活动进程，而且它将接管选择过程。接收者接管选择过程，除非它已经接管一个选择；
- 最后，其他进程都放弃只剩一个进程时，该进程成为新的协调者，然后它向所有的进程发送消息，通知这些进程从此刻起，它是新的协调者。

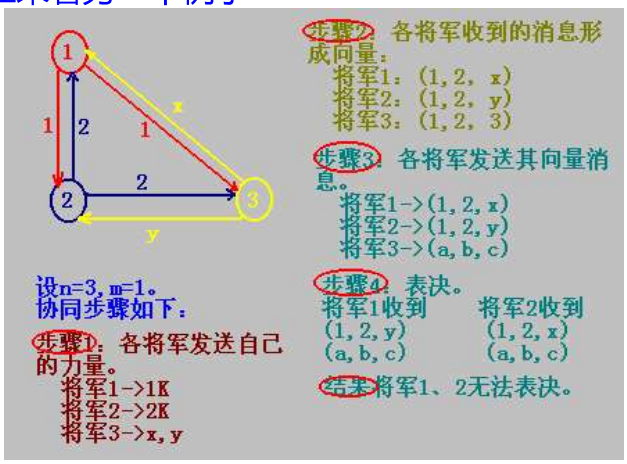
- 环算法(Ring Algorithm)
  - 对所有进程进行物理或逻辑的排序，这样每个进程都知道它的下一个进程是谁。
  - 任何一个进程发现协调者不起作用时，就创建一个包含该进程号(优先级)的Election消息，并将该消息发送给它的下一个(即后继)进程；
  - 如果后继进程已经中止，就跳过它发送给下一个，依此类推，直到找到一个活动进程为止；
  - 每个进程收到Election消息时，就将自己的进程号(和优先级)加入到Election消息的进程表中；
  - 最后Election消息会回到创建它的进程(通过判断进程表中是否含的自己的进程号),它从进程表中选择最大进程号(或优先级)的进程作为新的协调者，并构造一个Coordinator消息通知所有其他进程。

- 2.k-容错技术
  - 如果系统允许有k个组件出错，而仍能正常工作，这种容错技术称为k-容错技术。
  - 那么，Fail-silent faults须提供k+1个备份。Byzantine Faults则须提供2k+1个备份。
  - “两支军队”问题(two-army problem)
    - 说明即使两个处理器是正常的，但它们要达成一致的意见是困难的。
    - 一支红色军队，5000人，驻在山谷。两支蓝色的军队，每支3000人，驻扎在山角监视着山谷。两支蓝色的军队只有同时向红色军队发动进攻，才能获胜，任何一支单独出击都将失败。现在，两支蓝色军队的目标就是对进攻的时间达成一致的意见，然而他们的协同只能通过互派信使来交换意见，但信使随时可能被俘虏。
    - 假设一支蓝色军队的将军Alxander，决定在第二天凌晨2：00进攻，并派出一个信使将这个决定通知另一支蓝色军队的将军Bonaparte,之后就下达自己的军队，准备第二天凌晨2：00进攻；
    - 然而，到了第二于凌晨，Alexander担心，Bonaparte不知道信使已经安全返回，可能不会轻易进攻，结果，Alexander又派一个信使把确认的消息传给Bonaparte；
    - 假如信使也安全到达Bonaparte，这时，轮到Bonparte担心，Alexander不知道确认是否安全到达，可能不会按时进攻。这样Bonaparte只好也派一个信使将确认通知Alexander。
    - 这样，两个将军将来回确认，永远无法到成协议。因为最后一个发送消息的人不知道消息为对方接收，如果最后一个消息丢失，则所有协议都无效，无法进攻，并进入死循环。
    - 问题是：通信是不可靠的，下列我们假设通信是完善可靠的，仅仅是处理器可能出错。
  - Byzantine generals problem
    - 一支红色军队安营在山谷中，有n个将军率领n支蓝色军队扎营在山下周围，他们可以点对点完善地通信。假设其中有m(m<n)个将军可能是是背叛者(出错)，并且阻碍军队达成一致的意见(修改消息)，现在问题是正规军(非 背叛的)能否达成协同?

- 不失一般性，我们把这里的协同的含义稍作修改，即假设每个将军都掌握自己的军队的力量，问题的目标是各个将军能交换他们的部队力量，这样，算法结束时，每个将军就有一个向量表示盟军的力量， 如果将军是正规军，用i表示他的军队的力量，否则没有定义。

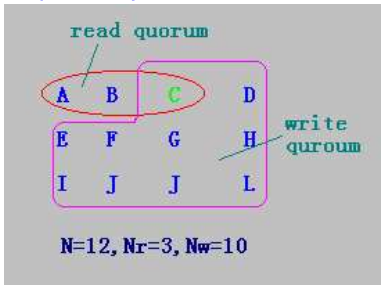


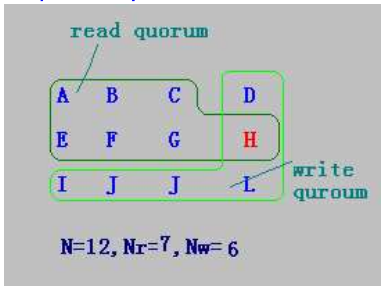
- 现在来看另一个例子

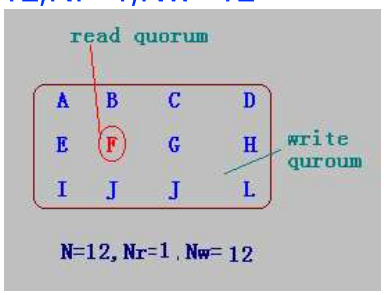


- 1982年，Lamport证明在一个系统中，有m个出错的处理器时，协同形成需要2m+1个正常工作的处理器，即总数是3m+1。协同形成需要约三分之二的人通过。

- 3.表决算法(voting algorithm)
  - 表决技术(votings)
    - 基本思想（1979，Gifford)
      - 客户读或写一个备份文件的操作前，先向所有的服务器发送请求，并得到多数服务器的许可后才能进行。
      - 设备份服务器有N个，并规定客户对备份文件的更新要经多数服务器的许可，即至少(N+1)/2个服务器的许可后，存取工作才可以进行，复制文件并形成新的版本（具有一版本号，对新修改的备份文件其版本号应该是相同的。）
      - “许可”，可以是一个含有版本号的应答消息。
      - 这样，读一个备份文件的步骤是：
        - 向至少半数以上的服务器([N+1]/2）发送读操作请求消息，要求返回文件的版本号
        - 如果所有的返回版本号是一致的，则说明该版本号是最近的，文件是最新的。因为其余的服务器个数不足半数，如果他们的版本号高，则说明其更新操作是无效的。如果更低，则说明上一次访问操作已经确认，只是这些服务器尚未更新。

        - 例如，有5个服务器，某客户得到3个服务器的版本号是8，则其余的两个服务器的版本号不可能是9，因为版本号从8到9要得到半数以上的服务器的许多，而不是2个。
    - 表决算法(voting algorithm)
        - 基本思想：某文件系统有N个备份服务器，定义Nr和Nw分别表示读法定人数(read quorum)和写法定人数(write quorum)：
            - 当客户的一个读操作得到Nr个服务器的相同版本号的许多时，可执行该读操作；
            - 当客户的一个写操作从Nw个服务器中得到相同的版本号的许可时，则可执行该写操作。
            - 其中Nr+Nw>N。
        - 例子
            - N=12,Nr=3,Nw=10
            - 

            - N=12,Nr=7,Nw=6
            - 

            - N=12,Nr=1,Nw=12
            - 

            - 特别地，Nr=1,Nw=12,意味着写操作要求所有的服务器表决，则读操作可以从中任选一个。
        - 说明
            - 由于读操作比写操作更经常发生，为提高算法效率，可规定Nr<Nw。
            - 当Nw接近N时，只要少数服务器的关闭，写操作就不能允许。

==========