Mutual Exclusion in Distributed systems

Assumptions

- N Independent Nodes/Sites
- Communication is only through message passing
- Messages ae delivered in finite time
- Communication delays are unknown and variable

Requirements

- **No Deadlock:**
  Two or more site should not endlessly wait for any message that will never arrive.
- **No Starvation:**
  Every site who wants to execute critical section should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute critical section while other site are repeatedly executing critical section
- **Fairness:**
  Each site should get a fair chance to execute critical section. Any request to execute critical section must be executed in the order they are made i.e., Critical section execution requests should be executed in the order of their arrival in the system.
- **Fault Tolerance:**
  In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

Logical Clock

- **Used for Ordering of events**
- **Logical Clocks** refer to implementing a protocol on all machines within your distributed system, so that the machines are able to maintain consistent ordering of events within some virtual timespan.
- A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system.
- Distributed systems may have no physically synchronous global clock, so a logical clock allows global ordering on events from different processes in such systems.

Timestamp

- Only local clock which is not synchronized
- Causality – Ordering of Events
  - Happen Before relationship
    - Taking single PC only if 2 events A and B are occurring one by one then TS(A) < TS(B). If A has timestamp of 1, then B should have timestamp more than 1, then only **happen before** relationship occurs.
    - Taking 2 PCs and event A in P1 (PC.1) and event B in P2 (PC.2) then also the condition will be TS(A) < TS(B). Taking example- suppose you are sending message to someone at 2:00:00 pm, and the other person is receiving it at 2:00:02 pm. Then it's obvious that TS(sender) < TS(receiver).
  - **Properties Derived from Happen Before Relationship –**
    - **Transitive Relation –**
      If, TS(A) <TS(B) and TS(B) <TS(C), then TS(A) < TS(C)
    - **Causally Ordered Relation –**
      a->b, this means that a is occurring before b and if there is any changes in a it will surely reflect on b.
    - **Concurrent Event –**
      This means that not every process occurs one by one, some processes are made to happen simultaneously i.e., A || B.

Timestamp Algorithm

1. A process increments its counter before each local event (e.g., message sending event);
2. When a process sends a message, it includes its counter value with the message after executing step 1;
3. On receiving a message, the counter of the recipient is updated, if necessary, to the greater of its current counter and the timestamp in the received message. The counter is then incremented by 1 before the message is considered received.

Ricart/Agrawala Algorithm

- Two type of messages (**REQUEST** and **REPLY**) are used and communication channels are assumed to follow FIFO order.
- A site sends a **REQUEST** message to all other site to get their permission to enter critical section.
- A site send a **REPLY** message to other site to give its permission to enter the critical section.
- A timestamp is given to each critical section request using Logical Timestamp.
- Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section request is always in the order of their timestamp.

Algorithm

- **To enter Critical section:**
  - When a site S_i wants to enter the critical section, it send a timestamped **REQUEST** message to all other sites.
  - When a site S_j receives a **REQUEST** message from site S_i, It sends a **REPLY** message to site S_i if and only if
    - Site S_j is neither requesting nor currently executing the critical section.
    - In case Site S_j is requesting and the timestamp of Site S_i's request is smaller than its own request.
  - Otherwise, the request is deferred by site S_j.

Algorithm

- **To execute the critical section:**
  - Site S_i enters the critical section if it has received the **REPLY** message from all other sites.
- **To release the critical section:**
  - Upon exiting site S_i sends **REPLY** message to all the deferred requests.

- Complexity – 2(n-1) messages

Maekawa's Algorithm

- Three type of messages (**REQUEST**, **REPLY** and **RELEASE**) are used.
- A site send a **REQUEST** message to all other site in its request set or quorum to get their permission to enter critical section.
- A site send a **REPLY** message to requesting site to give its permission to enter the critical section.
- A site send a **RELEASE** message to all other site in its request set or quorum upon exiting the critical section.

The construction of request set or Quorum:

1. $\forall i\ \forall j : i \neq j, 1 \leq i, j \leq N :: R_i \cap R_j \neq \emptyset$

i.e there is at least one common site between the request sets of any two sites.

2. $\forall i : 1 \leq i \leq N :: S_i \in R_i$
3. $\forall i : 1 \leq i \leq N :: |R_i| = K$
4. Any site S_i is contained in exactly K sets.
5. $N = K(K - 1) + 1$ and $|R_i| = \sqrt{N}$

Algorithm

- **To enter Critical section:**
  - When a site S_i wants to enter the critical section, it sends a request message **REQUEST(i)** to all other sites in the request set $\mathbf{R_i}$.
  - When a site S_j receives the request message **REQUEST(i)** from site S_i, it returns a **REPLY** message to site S_i if it has not sent a **REPLY** message to the site from the time it received the last **RELEASE** message. Otherwise, it queues up the request.
- .
- **To execute the critical section:**
  - A site S_i can enter the critical section if it has received the **REPLY** message from all the site in request set $\mathbf{R_i}$
- **To release the critical section:**
  - When a site S_i exits the critical section, it sends **RELEASE(i)** message to all other sites in request set $\mathbf{R_i}$
  - When a site S_j receives the **RELEASE(i)** message from site S_i, it send **REPLY** message to the next site waiting in the queue and deletes that entry from the queue
  - In case queue is empty, site S_j update its status to show that it has not sent any **REPLY** message since the receipt of the last **RELEASE** message

Maekawa’s Algorithm

- **Message Complexity:**
  Maekawa’s Algorithm requires invocation of 3√N messages per critical section execution as the size of a request set is √N. These 3√N messages involves.
  - √N request messages
  - √N reply messages
  - √N release messages
- **Drawbacks of Maekawa’s Algorithm:**
  - This algorithm is deadlock prone because a site is exclusively locked by other sites and requests are not prioritized by their timestamp.
- **Performance:**
  - Synchronization delay is equal to twice the message propagation delay time
  - It requires 3√n messages per critical section execution.