Distributed Synchronization

Distributed Systems

A distributed system consists of a number of processes.

Each process has a state (Values of variables)

Each process takes action to change its state, which may be an instruction or a communication action (send, receive)

- Each process has a common clock and events within a process can be assigned timestamps and thus ordered.
- In DS, the time order of events across different processes are also to be considered.
- The multiple processors do not share a common memory or clock.
- Instead, each processor has its own local memory and communicates with other processes through communication lines.

The Importance of Synchronization

- Various components of a distributed system must cooperate and exchange information, synchronization is a necessity.
- Constraints, both implicit and explicit, are therefore enforced to ensure synchronization of components.
- Synchronization is required for
  - Fairness
  - Correctness

Clock Synchronization

- Logical Clocks
  - Happened Before relation
- Physical Clocks
  - Centralized
    - Broadcast Based
    - Request Driven
  - Decentralized

Physical Clocks & Synchronization

- In a DS, each process has its own clock.
- Clock Skew versus Drift
  - Clock Skew = Relative Difference in clock *values* of two processes
  - Clock Drift = Relative Difference in clock *frequencies (rates)* of two processes
- *A non-zero clock drift will cause skew to continuously increase.*

Implementation of Physical Time Service

- Obtaining accurate value when implmenting a physical time service
- Synchronizing the concept of physical time throughout the distributed system.
- Implementation
  - Centralized
  - Distributed

Centralized Physical service

- Broadcast based
- Request driven

Problems

- Single point of failure
- Traffic around server increases
- Not scalable

Distributed Physical service

- Client broadcast its current time at predefined set intervals
- Starts timer
- Collects messages
- Calculates average values and then adjust time values.

Process Synchronization

Techniques to coordinate execution among processes

- One process may have to wait for another
- Shared resource (e.g. critical section) may require exclusive access

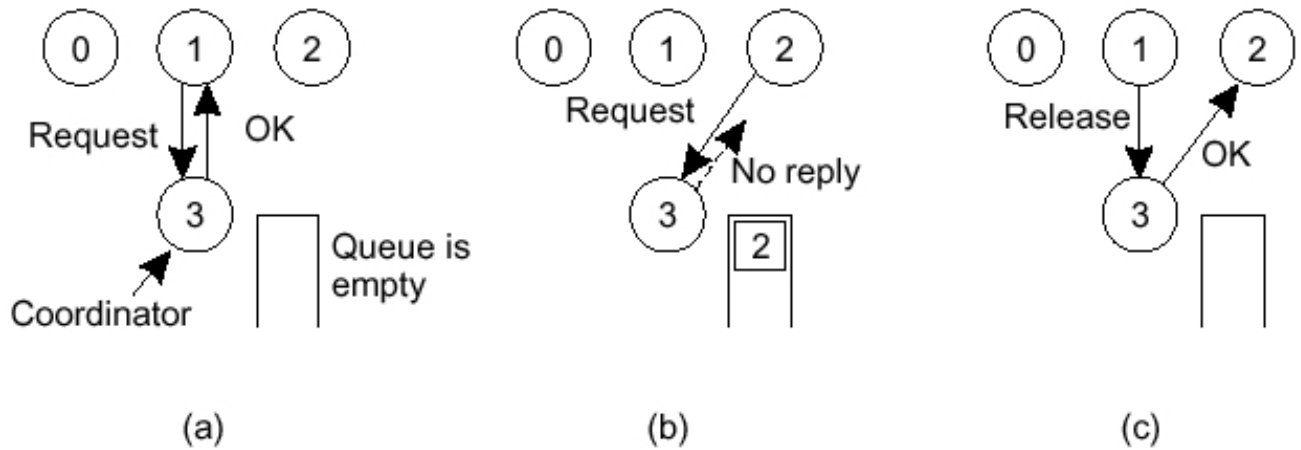
Mutual Exclusion

- Centralized

- Distributed
  - Lock-based (aka permission based, non-token based) - to enter the CS a process needs to obtain permission from other processes in the system.
  - Token-based - unique token (privilege) circulated in the system. A process possessing the token can enter CS

Centralized Algorithm

- One process is elected as the coordinator.
- When any process wants to enter a critical section, it sends a request message to the coordinator stating which critical section it wants to access.
- If no other process is currently in that critical section, the coordinator sends back a reply granting permission. When the reply arrives, the requesting process enters the critical section. If another process requests access to the same critical section, it is ignored or blocked until the first process exits the critical section and sends a message to the coordinator stating that it has exited.


0
1
2
Request
OK
3
Coordinator
Queue is
empty
(a)
0
1
2
Request
No reply
3
2
(b)
0
1
2
Release
OK
3
(c)

Distributed Algorithm – Permission based

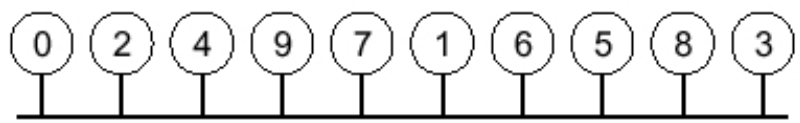
- When a process wants to enter a critical section, it builds a message containing the name of the critical section, its process number, and the current time. It then sends the message to all other processes, as well as to itself.

- When a process receives a request message, the action it takes depends on its state with respect to the critical section named in the message. There are three cases: If the receiver is not in the critical section and does not want to enter it, it sends an OK message to the sender.
- If the receiver is in the critical section, it does not reply. It instead queues the request.
- If the receiver also wants to enter the same critical section, it compares the time stamp in the incoming message with the time stamp in the message it has sent out. The lowest time stamp wins. If its own message has a lower time stamp, it does not reply and queues the request from the sending process.
- When a process has received OK messages from all other processes, it enters the critical section. Upon exiting the critical section, it sends OK messages to all processes in its queue and deletes them all from the queue.

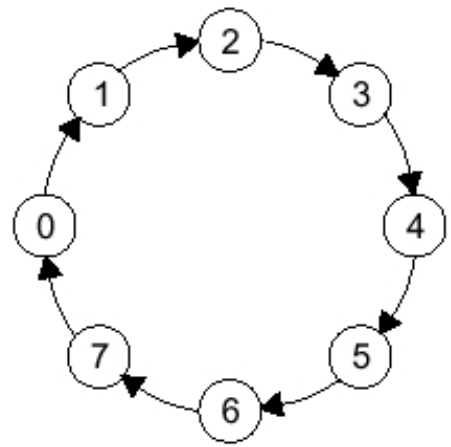
Token Based Algorithm

- Another approach is to create a logical or physical ring.
- Each process knows the identity of the process succeeding it.
- When the ring is initialized, Process 0 is given a token. The token circulates around the ring in order, from Process k to Process k + 1.
- When a process receives the token from its neighbor, it checks to see if it is attempting to enter a critical section. If so, the process enters the critical section and does its work, keeping the token the whole time.

- After the process exits the critical section, it passes the token to the next process in the ring. It is not permitted to enter a second critical section using the same token.
- If a process is handed a token an is not interested in entering a critical section, it passes the token to the next process.



(a)



(b)