UNIT III

DISTRIBUTED MUTEX & DEADLOCK

DISTRIBUTED MUTEX & DEADLOCK

Distributed mutual exclusion algorithms: Introduction – Preliminaries – Lamport‘s algorithm –Ricart-Agrawala algorithm – Maekawa‘s algorithm – Suzuki–Kasami‘s broadcast algorithm. Deadlock detection in distributed systems: Introduction – System model – Preliminaries –Models of deadlocks – Knapp‘s classification – Algorithms for the single resource model, the AND model and the OR model.

3.1 DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS

- Mutual exclusion is a concurrency control property which is introduced to prevent race conditions.
- It is the requirement that a process cannot access a shared resource while another concurrent process is currently present or executing the same resource.

Mutual exclusion in a distributed system states that only one process is allowed to execute the critical section (CS) at any given time.

- Message passing is the sole means for implementing distributed mutual exclusion.
- The decision as to which process is allowed access to the CS next is arrived at by message passing, in which each process learns about the state of all other processes in some consistent way.
- There are three basic approaches for implementing distributed mutual exclusion:

1. Token-based approach:

- A unique token is shared among all the sites.
- If a site possesses the unique token, it is allowed to enter its critical section
- This approach uses sequence number to order requests for the critical section.
- Each requests for critical section contains a sequence number. This sequence number is used to distinguish old and current requests.
- This approach insures Mutual exclusion as the token is unique.
- Eg: Suzuki-Kasami’s Broadcast Algorithm

2. Non-token-based approach:

- A site communicates with other sites in order to determine which sites should execute critical section next. This requires exchange of two or more successive round of messages among sites.
- This approach use timestamps instead of sequence number to order requests for the critical section.
- When ever a site make request for critical section, it gets a timestamp. Timestamp is also used to resolve any conflict between critical section requests.
- All algorithm which follows non-token based approach maintains a logical clock. Logical clocks get updated according to Lamport’s scheme.
- Eg: Lamport's algorithm, Ricart–Agrawala algorithm

3. Quorum-based approach:

- Instead of requesting permission to execute the critical section from all other sites, Each site requests only a subset of sites which is called a quorum.
- Any two subsets of sites or Quorum contains a common site.
- This common site is responsible to ensure mutual exclusion.
- Eg: Maekawa's Algorithm

3.1.1 Preliminaries

- The system consists of N sites, S_1, S_2, S_3, …, S_N.
- Assume that a single process is running on each site.
- The process at site S_i is denoted by p_i. All these processes communicate asynchronously over an underlying communication network.
- A process wishing to enter the CS requests all other or a subset of processes by sending REQUEST messages, and waits for appropriate replies before entering the CS.
- While waiting the process is not allowed to make further requests to enter the CS.
- A site can be in one of the following three states: requesting the CS, executing the CS, or neither requesting nor executing the CS.
- In the requesting the CS state, the site is blocked and cannot make further requests for the CS.
- In the idle state, the site is executing outside the CS.
- In the token-based algorithms, a site can also be in a state where a site holding the token is executing outside the CS. Such state is referred to as the idle token state.
- At any instant, a site may have several pending requests for CS. A site queues up these requests and serves them one at a time.
- N denotes the number of processes or sites involved in invoking the critical section, T denotes the average message delay, and E denotes the average critical section execution time.

3.1.2 Requirements of mutual exclusion algorithms

- **Safety property:**

The safety property states that at any instant, only one process can execute the critical section. This is an essential property of a mutual exclusion algorithm.

- **Liveness property:**

This property states the absence of deadlock and starvation. Two or more sites should not endlessly wait for messages that will never arrive. In addition, a site must not wait indefinitely to execute the CS while other sites are repeatedly executing the CS. That is, every requesting site should get an opportunity to execute the CS in finite time.

- **Fairness:**

Fairness in the context of mutual exclusion means that each process gets a fair chance to execute the CS. In mutual exclusion algorithms, the fairness property generally means that the CS execution requests are executed in order of their arrival in the system.

3.1.3 Performance metrics

- **Message complexity:** This is the number of messages that are required per CS execution by a site.
- **Synchronization delay:** After a site leaves the CS, it is the time required and before the next site enters the CS. (Figure 3.1)
- **Response time:** This is the time interval a request waits for its CS execution to be over after its request messages have been sent out. Thus, response time does not include the time a request waits at a site before its request messages have been sent out. (Figure 3.2)
- **System throughput:** This is the rate at which the system executes requests for the CS. If SD is the synchronization delay and E is the average critical section execution time.

$$\text{System throughput} = \frac{1}{(SD+E)}$$

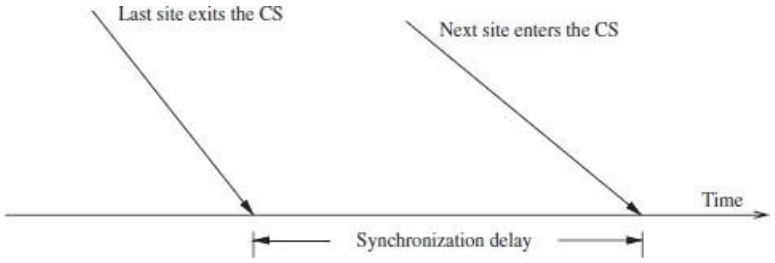

Figure 3.1 Synchronization delay

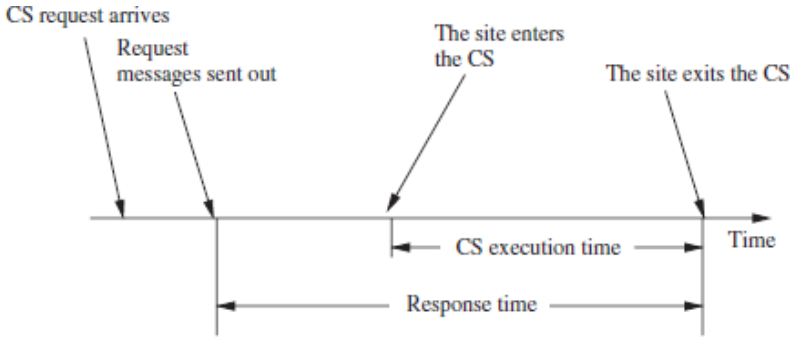

Figure 3.2 Response Time

Low and High Load Performance:

- The performance of mutual exclusion algorithms is classified as two special loading conditions, viz., “low load” and “high load”.
- The load is determined by the arrival rate of CS execution requests.
- Under ***low load*** conditions, there is seldom more than one request for the critical section present in the system simultaneously.
- Under ***heavy load*** conditions, there is always a pending request for critical section at a site.

Best and worst case performance

- In the best case, prevailing conditions are such that a performance metric attains the best possible value. For example, the best value of the response time is a roundtrip message delay plus the CS execution time, 2T +E.

- For examples, the best and worst values of the response time are achieved when load is, respectively, low and high;
- The best and the worse message traffic is generated at low and heavy load conditions, respectively.

3.2 LAMPORT'S ALGORITHM

- Lamport's Distributed Mutual Exclusion Algorithm is a permission based algorithm proposed by Lamport as an illustration of his synchronization scheme for distributed systems.
- In permission based timestamp is used to order critical section requests and to resolve any conflict between requests.
- In Lamport's Algorithm critical section requests are executed in the increasing order of timestamps i.e a request with smaller timestamp will be given permission toexecute critical section first than a request with larger timestamp.
- Three type of messages (REQUEST, REPLY and RELEASE) are used and communication channels are assumed to follow FIFO order.
- A site send a REQUEST message to all other site 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 upon exiting the critical section.
- Every site Si, keeps a queue to store critical section requests ordered by their timestamps.
- request_queuei denotes the queue of site Si.
- A timestamp is given to each critical section request using Lamport's logical clock.
- 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.

Requesting the critical section

- When a site S_i wants to enter the CS, it broadcasts a REQUEST(ts_i, i) message to all other sites and places the request on $request\_queue_i$. ((ts_i, i) denotes the timestamp of the request.)
- When a site S_j receives the REQUEST(ts_i, i) message from site S_i, it places site S_i's request on $request\_queue_j$ and returns a timestamped REPLY message to S_i.

Executing the critical section

Site S_i enters the CS when the following two conditions hold:

L1: S_i has received a message with timestamp larger than (ts_i, i) from all other sites.

L2: S_i's request is at the top of $request\_queue_i$.

Releasing the critical section

- Site S_i, upon exiting the CS, removes its request from the top of its request queue and broadcasts a timestamped RELEASE message to all other sites.
- When a site S_j receives a RELEASE message from site S_i, it removes S_i's request from its request queue.

Fig 3.1: Lamport's distributed mutual exclusion algorithm

To enter Critical section:

- When a site S_i wants to enter the critical section, it sends a request message Request(ts_i, i) to all other sites and places the request on $request\_queue_i$. Here, T_{si} denotes the timestamp of Site S_i.
- When a site S_j receives the request message REQUEST(ts_i, i) from site Si, it returns a timestamped REPLY message to site Si and places the request of site Si on $request\_queue_j$

To execute the critical section:

- A site S_i can enter the critical section if it has received the message with timestamp larger than (ts_i, i) from all other sites and its own request is at the top of $request\_queue_i$.

To release the critical section:

- When a site S_i exits the critical section, it removes its own request from the top of its request queue and sends a timestamped RELEASE message to all other sites. When a site S_j receives the timestamped RELEASE message from site S_i, it removes the request of S_{ia} from its request queue.

Correctness

Theorem: Lamport's algorithm achieves mutual exclusion.

Proof: Proof is by contradiction.

- Suppose two sites S_i and S_j are executing the CS concurrently. For this to happen conditions L1 and L2 must hold at both the sites concurrently.
- This implies that at some instant in time, say t, both S_i and S_j have their own requests at the top of their request queues and condition L1 holds at them. Without loss of generality, assume that S_i 's request has smaller timestamp than the request of S_j .
- From condition L1 and FIFO property of the communication channels, it is clear that at instant t the request of S_i must be present in $request\ queue_j$ when S_j was executing its CS. This implies that Sj 's own request is at the top of its own request queue whena smaller timestamp request, Si 's request, is present in the $request\ queue_j$ – a contradiction!

Theorem: Lamport's algorithm is fair.

***Proof: The proof is by contradiction*.**

- Suppose a site S_i 's request has a smaller timestamp than the request of another site S_j and S_j is able to execute the CS before S_i .
- For S_j to execute the CS, it has to satisfy the conditions L1 and L2. This implies that at some instant in time say t, S_j has its own request at the top of its queue and it has also received a message with timestamp larger than the timestamp of its request from all other sites.
- But request queue at a site is ordered by timestamp, and according to our assumption Si has lower timestamp. So S_i 's request must be placed ahead of the S_j 's request in the $request\ queue_j$. This is a contradiction!

Message Complexity:

Lamport's Algorithm requires invocation of 3(N – 1) messages per critical section execution. These 3(N – 1) messages involves

- (N – 1) request messages
- (N – 1) reply messages
- (N – 1) release messages

Drawbacks of Lamport's Algorithm:

- **Unreliable approach**: failure of any one of the processes will halt the progress of entire system.
- **High message complexity:** Algorithm requires 3(N-1) messages per critical section invocation.

Performance:

Synchronization delay is equal to maximum message transmission time. It requires 3(N – 1) messages per CS execution. Algorithm can be optimized to 2(N – 1) messages by omitting the REPLY message in some situations.