Introduction

- Mutual exclusion: Concurrent access of processes to a shared resource or data is executed in mutually exclusive manner.
- Only one process is allowed to execute the critical section (CS) at any given time.
- In a distributed system, shared variables (semaphores) or a local kernel cannot be used to implement mutual exclusion.
- Message passing is the sole means for implementing distributed mutual exclusion.

Introduction

- Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.
- Three basic approaches for distributed mutual exclusion:
  1. Token based approach
  2. Non-token based approach
  3. Quorum based approach
- Token-based approach:
  - A unique token is shared among the sites.
  - A site is allowed to enter its CS if it possesses the token.
  - Mutual exclusion is ensured because the token is unique.

Introduction

- Non-token based approach:
  - Two or more successive rounds of messages are exchanged among the sites to determine which site will enter the CS next.
- Quorum based approach:
  - Each site requests permission to execute the CS from a subset of sites (called a quorum).
  - Any two quorums contain a common site.
  - This common site is responsible to make sure that only one request executes the CS at any time.

Preliminaries

System Model

- The system consists of N sites, S_1, S_2, ..., S_N.
- We assume that a single process is running on each site. The process at site S_i is denoted by p_i.
- A site can be in one of the following three states: requesting the CS, executing the CS, or neither requesting nor executing the CS (i.e., idle).
- In the 'requesting the CS' state, the site is blocked and can not make further requests for the CS. In the 'idle' state, the site is executing outside the CS.
- In token-based algorithms, a site can also be in a state where a site holding the token is executing outside the CS (called 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.

Requirements

Requirements of Mutual Exclusion Algorithms

1. **Safety Property:** At any instant, only one process can execute the critical section.
2. **Liveness Property:** This property states the absence of deadlock and starvation. Two or more sites should not endlessly wait for messages which will never arrive.
3. **Fairness:** Each process gets a fair chance to execute the CS. Fairness property generally means the CS execution requests are executed in the order of their arrival (time is determined by a logical clock) in the system.

Performance Metrics

The performance is generally measured by the following four metrics:

- **Message complexity:** The number of messages 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 (see Figure 1).

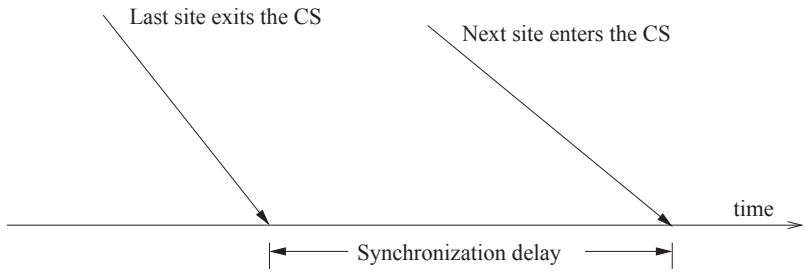

Figure 1: Synchronization Delay.

Performance Metrics

- **Response time:** The time interval a request waits for its CS execution to be over after its request messages have been sent out (see Figure 2).

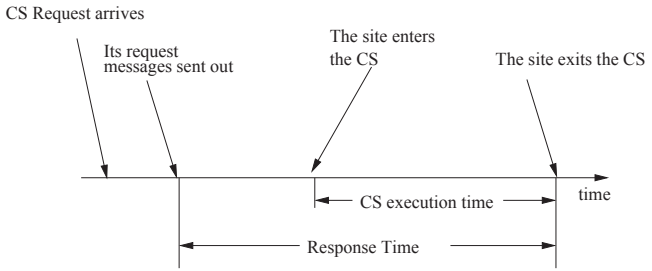

Figure 2: Response Time.

- **System throughput:** The rate at which the system executes requests for the CS.

$$\text{system throughput} = 1/(SD+E)$$

where SD is the synchronization delay and E is the average critical section execution time.

Performance Metrics

Low and High Load Performance:

- We often study the performance of mutual exclusion algorithms under 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.

Lamport's Algorithm

- Requests for CS are executed in the increasing order of timestamps and time is determined by logical clocks.
- Every site S_i keeps a queue, $request\_queue_i$, which contains mutual exclusion requests ordered by their timestamps.
- This algorithm requires communication channels to deliver messages the FIFO order.

The Algorithm

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,places site S_i's request on $request\_queue_j$ and it 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 $request\_queue_i$.

The Algorithm

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.

When a site removes a request from its request queue, its own request may come at the top of the queue, enabling it to enter the CS.

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 S_j's own request is at the top of its own *request_queue* when a smaller timestamp request, S_i's request, is present in the $request\_queue_j$ – a contradiction!

correctness

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 S_i 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!

Performance

- For each CS execution, Lamport's algorithm requires $(N-1)$ REQUEST messages, $(N-1)$ REPLY messages, and $(N-1)$ RELEASE messages.
- Thus, Lamport's algorithm requires $3(N-1)$ messages per CS invocation.
- Synchronization delay in the algorithm is T.

An optimization

- In Lamport's algorithm,REPLY messages can be omitted in certain situations. For example, if site S_j receives a REQUEST message from site S_i after it has sent its own REQUEST message with timestamp higher than the timestamp of site S_i's request, then site S_j need not send a REPLY message to site S_i.
- This is because when site S_i receives site S_j's request with timestamp higher than its own, it can conclude that site S_j does not have any smaller timestamp request which is still pending.
- With this optimization, Lamport's algorithm requires between $3(N-1)$ and $2(N-1)$ messages per CS execution.

Token-Based Algorithms

- In token-based algorithms, a unique token is shared among the sites.
- A site is allowed to enter its CS if it possesses the token.
- Token-based algorithms use sequence numbers instead of timestamps. (Used to distinguish between old and current requests.)

Suzuki-Kasami's Broadcast Algorithm

- If a site wants to enter the CS and it does not have the token, it broadcasts a REQUEST message for the token to all other sites.
- A site which possesses the token sends it to the requesting site upon the receipt of its REQUEST message.
- If a site receives a REQUEST message when it is executing the CS, it sends the token only after it has completed the execution of the CS.

continuation..

This algorithm must efficiently address the following two design issues:
(1) How to distinguish an outdated REQUEST message from a current REQUEST message:

- Due to variable message delays, a site may receive a token request message after the corresponding request has been satisfied.
- If a site can not determined if the request corresponding to a token request has been satisfied, it may dispatch the token to a site that does not need it.
- This will not violate the correctness, however, this may seriously degrade the performance.

(2) How to determine which site has an outstanding request for the CS:

- After a site has finished the execution of the CS, it must determine what sites have an outstanding request for the CS so that the token can be dispatched to one of them.

continuation..

The first issue is addressed in the following manner:

- A REQUEST message of site S_j has the form REQUEST(j, n) where n (n=1, 2, ...) is a sequence number which indicates that site S_j is requesting its n^{th} CS execution.
- A site S_i keeps an array of integers RN_i[1..N] where RN_i[j] denotes the largest sequence number received in a REQUEST message so far from site S_j.
- When site S_i receives a REQUEST(j, n) message, it sets RN_i[j]:= max(RN_i[j], n).
- When a site S_i receives a REQUEST(j, n) message, the request is outdated if RN_i[j]>n.

continuation..

The second issue is addressed in the following manner:

- The token consists of a queue of requesting sites, Q, and an array of integers LN[1..N], where LN[j] is the sequence number of the request which site S_j executed most recently.
- After executing its CS, a site S_i updates LN[i]:=RN_i[i] to indicate that its request corresponding to sequence number RN_i[i] has been executed.
- At site S_i if RN_i[j]=LN[j]+1, then site S_j is currently requesting token.

The Algorithm

Requesting the critical section

(a) If requesting site S_i does not have the token, then it increments its sequence number, RN_i[i], and sends a REQUEST(i, sn) message to all other sites. ('sn' is the updated value of RN_i[i].)

(b) When a site S_j receives this message, it sets RN_j[i] to max(RN_j[i], sn). If S_j has the idle token, then it sends the token to S_i if RN_j[i]=LN[i]+1.

Executing the critical section

(c) Site S_i executes the CS after it has received the token.

The Algorithm

Releasing the critical section Having finished the execution of the CS, site S_i takes the following actions:

(d) It sets LN[i] element of the token array equal to RN_i[i].

(e) For every site S_j whose id is not in the token queue, it appends its id to the token queue if RN_i[j]=LN[j]+1.

(f) If the token queue is nonempty after the above update, S_i deletes the top site id from the token queue and sends the token to the site indicated by the id.

Correctness

Mutual exclusion is guaranteed because there is only one token in the system and a site holds the token during the CS execution.

Theorem: A requesting site enters the CS in finite time.

Proof:

- Token request messages of a site S_i reach other sites in finite time.
- Since one of these sites will have token in finite time, site S_i's request will be placed in the token queue in finite time.
- Since there can be at most $N-1$ requests in front of this request in the token queue, site S_i will get the token and execute the CS in finite time.

Performance

- No message is needed and the synchronization delay is zero if a site holds the idle token at the time of its request.
- If a site does not hold the token when it makes a request, the algorithm requires N messages to obtain the token. Synchronization delay in this algorithm is 0 or T.