M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Distributed Systems
Course Code: CSE20/CSE751
Credits: 3:0:0

Term: Oct 2021-Feb 2022

Faculty:
Sini Anna Alex

Correctness

Theorem: Ricart-Agrawala algorithm achieves mutual exclusion.

Proof:

- Proof is by contradiction. Suppose two sites S_i and S_j' are executing the CS concurrently and S_i's request has higher priority than the request of S_j. Clearly, S_i received S_j's request after it has made its own request.
- Thus, S_j can concurrently execute the CS with S_i only if S_i returns a REPLY to S_j (in response to S_j's request) before S_i exits the CS.
- However, this is impossible because S_j's request has lower priority. Therefore, Ricart-Agrawala algorithm achieves mutual exclusion.

Performance

- For each CS execution, Ricart-Agrawala algorithm requires $(N-1)$ REQUEST messages and $(N-1)$ REPLY messages.
- Thus, it requires $2(N-1)$ messages per CS execution.
- Synchronization delay in the algorithm is T.

RAMAIAH Institute of Technology

Singhal's Dynamic Information-Structure Algorithm

- Most mutual exclusion algorithms use a static approach to invoke mutual exclusion.
- These algorithms always take the same course of actions to invoke mutual exclusion no matter what is the state of the system.
- These algorithms lack efficiency because they fail to exploit the changing conditions in the system.
- An algorithm can exploit dynamic conditions of the system to improve the performance.

RAMAIAH
Institute of Technology

Singhal's Dynamic Information-Structure Algorithm

- For example, if few sites are invoking mutual exclusion very frequently and other sites invoke mutual exclusion much less frequently, then
  - A frequently invoking site need not ask for the permission of less frequently invoking site every time it requests an access to the CS.
  - It only needs to take permission from all other frequently invoking sites.
- Singhal developed an adaptive mutual exclusion algorithm based on this observation.
- The information-structure of the algorithm evolves with time as sites learn about the state of the system through messages.

System Model

- We consider a distributed system consisting of n autonomous sites, say, S_1, S_2, ..., S_n, connected by a communication network.
- We assume that the sites communicate completely by message passing.
- Message propagation delay is finite but unpredictable.
- Between any pair of sites, messages are delivered in the order they are sent.
- The underlying communication network is reliable and sites do not crash.

Data Structures

- Information-structure at a site S_i consists of two sets. The first set R_i, called *request set*, contains the sites from which S_i must acquire permission before executing CS.
- The second set I_i, called *inform set*,contains the sites to which S_i must send its permission to execute CS after executing its CS.
- Every site S_i maintains a logical clock C_i, which is updated according to Lamport's rules.
- Every site maintains three boolean variables to denote the state of the site: *Requesting, Executing, and My_priority*.
- Requesting and executing are true if and only if the site is requesting or executing CS, respectively. My_priority is true if pending request of S_i has priority over the current incoming request.

RAMAIAH
Institute of Technology

Initialization

The system starts in the following initial state:

For a site S_i (i = 1 to n),

$$R_i := \{S_1, S_2, \ldots, S_i - 1, S_i\}$$
$$I_i := \{S_i\}$$
$$C_i := 0$$
$$Requesting_i = Executing_i := \text{False}$$

Initialization

If we stagger sites S_n to S_1 from left to right, then the initial system state has the following two properties:

- For a site, only all the sites to its left will ask for its permission and it will ask for the permission of only all the sites to its right.
- The cardinality of R_i decreases in stepwise manner from left to right. Due to this reason, this configuration has been called "staircase pattern" in topological sense .

The Algorithm

Step 1: (Request Critical Section)

Requesting = true;

$C_i = C_i + 1$;

Send REQUEST(C_i, i) message to all sites in R_i;

Wait until $R_i = \emptyset$;

/* Wait until all sites in R_i have sent a reply to S_i */

Requesting = false;

Step 2: (Execute Critical Section)

Executing = true;

Execute CS;

Executing = false;

RAMAIAH Institute of Technology

The Algorithm

Step 3: (Release Critical Section)

For every site S_k in I_i (except S_i) do

Begin

$I_i = I_i - \{S_k\}$;

Send REPLY(C_i, i) message to S_k;

$R_i = R_i + \{S_k\}$

End

The Algorithm

REQUEST message handler

/* Site S_i is handling message REQUEST(c, j) */

$C_i := \max\{C_i, c\}$;

Case

Requesting = true:

Begin if My_priority then $I_i := I_i + \{j\}$

/*My_Priority true if pending request of S_i has priority over incoming request */

Else

Begin

Send REPLY(C_i, i) message to S_j;

If not ($S_j \in R_i$) then

Begin

$R_i = R_i + \{S_j\}$;

Send REQUEST(C_i, i) message to site S_j;

End;

End;

Executing = true: $I_i = I_i + \{S_j\}$;

Executing = false $\wedge$ **Requesting = false:**

Begin

$R_i = R_i + \{S_j\}$;

Send REPLY(C_i, i) message to S_j;

End;

An Explanation of the Algorithm

- S_i acquires permission to execute the CS from all sites in its request set R_i and it releases the CS by sending a REPLY message to all sites in its inform set I_i.
- If site S_i which itself is requesting the CS, receives a higher priority REQUEST message from a site S_j, then S_i takes the following actions:
  1. (i) S_i immediately sends a REPLY message to S_j,
  2. (ii) if S_j is not in R_i, then S_i also sends a REQUEST message to S_j, and
  3. (iii) S_i places an entry for S_j in R_i. Otherwise, S_i places an entry for S_j into I_i so that S_j can be sent a REPLY message when S_i finishes with the execution of the CS.
- If S_i receives a REQUEST message from S_j when it is executing the CS, then it simply puts S_j in I_i so that S_j can be sent a REPLY message when S_i finishes with the execution of the CS.
- If S_i receives a REQUEST message from S_j when it is neither requesting nor executing the CS, then it places an entry for S_j in R_i and sends S_j a REPLY message.

Thank you