CS 3551 DISTRIBUTED COMPUTING

UNIT III DISTRIBUTED MUTEX AND DEADLOCK 10

Distributed Mutual exclusion Algorithms: Introduction – Preliminaries – Lamport's algorithm – Ricart-Agrawala's Algorithm — Token-Based Algorithms – Suzuki-Kasami's Broadcast Algorithm; Deadlock Detection in Distributed Systems: Introduction – System Model – Preliminaries – Models of Deadlocks – Chandy-Misra-Haas Algorithm for the AND model and OR Model.

Ricart Agarwala Algorithm

Enter ?

1. **Can work in non-FIFO channel**
2. **Give chance to process with lowest request timestamp**

P1: When I want to enter the CS? => Send out REQUEST MSG to all

P2: What I do when REQUEST received?

Case 1: If I am not in CS or I havn't sent REQUEST to enter CS, then send REPLY.

Case 2: I want to enter but you have made request earlier, REPLY.

P_2 - 2pm
P_1 - 1.50pm

Case 3: Defer sending the reply and make $RD_2[3] = 1$

$P_1 \; P_2 \; P_3$ $\quad$ P_2 $\quad RD_2 \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$

Exit : Send all deferred replies.

$S_i \xrightarrow[req]{} S_j$

Requesting the critical section

(a) When a site S_i wants to enter the CS, it broadcasts a timestamped REQUEST message to all other sites.

(b) When site S_j receives a REQUEST message from site S_i, it sends a REPLY message to site S_i if site S_j is neither requesting nor executing the CS, or if the site S_j is requesting and S_i's request's timestamp is smaller than site S_j's own request's timestamp. Otherwise, the reply is deferred and S_j sets $RD_j[i] := 1$.

Executing the critical section

(c) Site S_i enters the CS after it has received a REPLY message from every site it sent a REQUEST message to.

Releasing the critical section

(d) When site S_i exits the CS, it sends all the deferred REPLY messages: $\forall j$ if $RD_i[j] = 1$, then sends a REPLY message to S_j and sets $RD_i[j] := 0$.

Algorithm 9.2 The Ricart–Agrawala algorithm.

Key Points

- A process sends a REQUEST message to all other processes to request their permission to enter the critical section.
- A process sends a REPLY message to a process to give its permission to that process. Processes use Lamport-style logical clocks to assign a timestamp to critical section requests.
- Timestamps are used to decide the priority of requests in case of conflict – if a process p_i that is waiting to execute the critical section receives a REQUEST message from process p_j, then if the priority of p_j's request is lower, p_i defers the REPLY to p_j and sends a REPLY message to p_j only after executing the CS for its pending request.
- Otherwise, p_i sends a REPLY message to p_j immediately, provided it is currently not executing the CS.
- Thus, if several processes are requesting execution of the CS, the highest priority request succeeds in collecting all the needed REPLY messages and gets to execute the CS.

Each process p_i maintains the request-deferred array, RD_i, the size of which is the same as the number of processes in the system. Initially, $\forall i\ \forall j$: $RD_i[j] = 0$. Whenever p_i defers the request sent by p_j, it sets $RD_i[j] = 1$, and after it has sent a REPLY message to p_j, it sets $RD_i[j] = 0$.

Key Points continued…..

- When a site receives a message, it updates its clock using the timestamp in the message.
- Also, when a site takes up a request for the CS for processing, it updates its local clock and assigns a timestamp to the request.
- In this algorithm, a site's REPLY messages are blocked only by sites that are requesting the CS with higher priority (i.e., smaller timestamp).
- Thus, when a site sends out deferred REPLY messages, the site with the next highest priority request receives the last needed REPLY message and enters the CS. Execution of the CS requests in this algorithm is always in the order of their timestamps.


thank you


LIKE
COMMENT
SHARE
SUBSCRIBE

Theorem : Algorithm achieves Mutual Exclusion

reply

$S_j \xrightarrow{\text{req}} S_i$

①

CS

S_i
S_j

② $S_{i_T} < S_{j_T}$

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 (i.e., smaller timestamp) than the request of S_j. Clearly, S_i received S_j's request after it has made its own request. (Otherwise, S_i's request will have lower priority.) 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, the Ricart–Agrawala algorithm achieves mutual exclusion. □

Figure 9.7 Sites S_1 and S_2 each make a request for the CS.

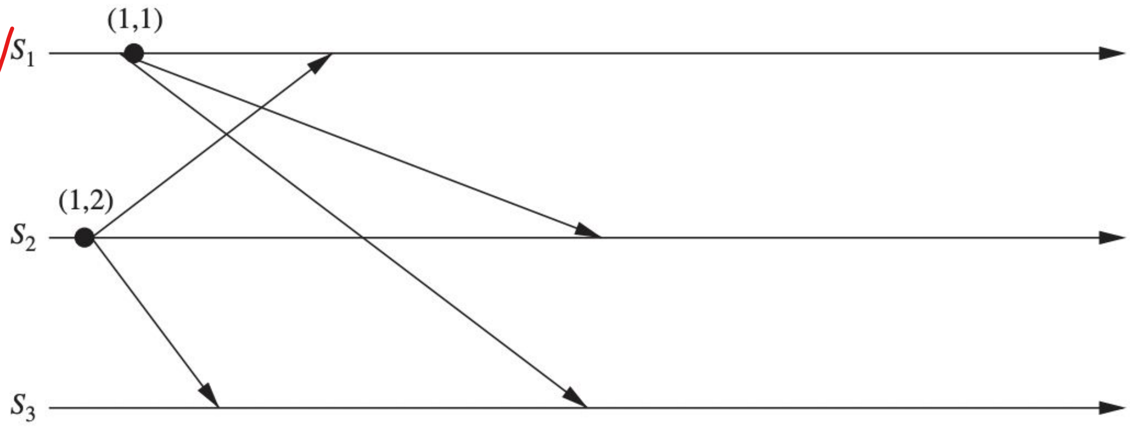


Figure 9.8 Site S_1 enters the CS.

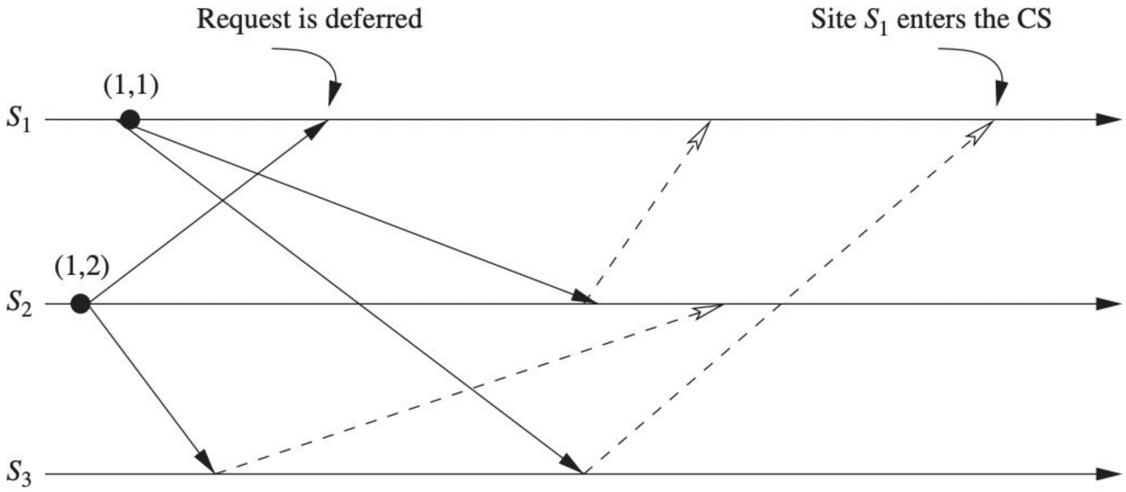


Figure 9.9 Site S_1 exits the CS and sends a REPLY message to S_2's deferred request.

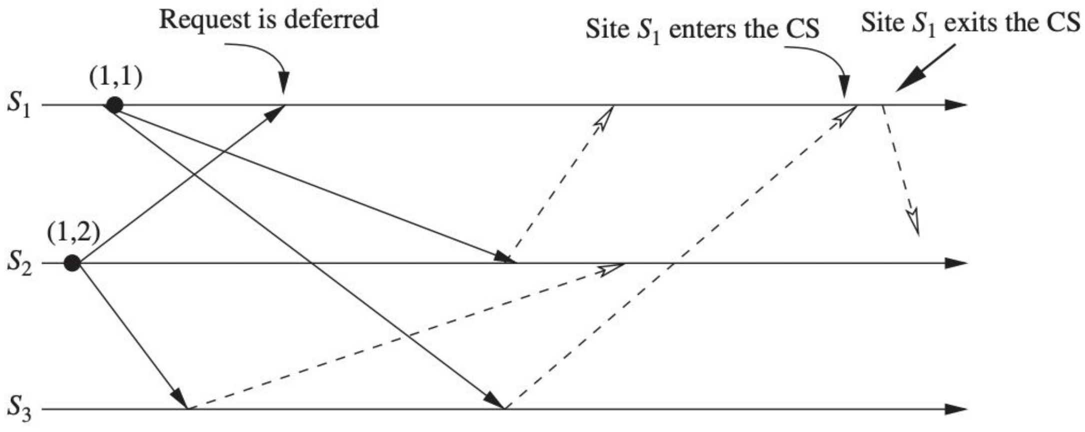
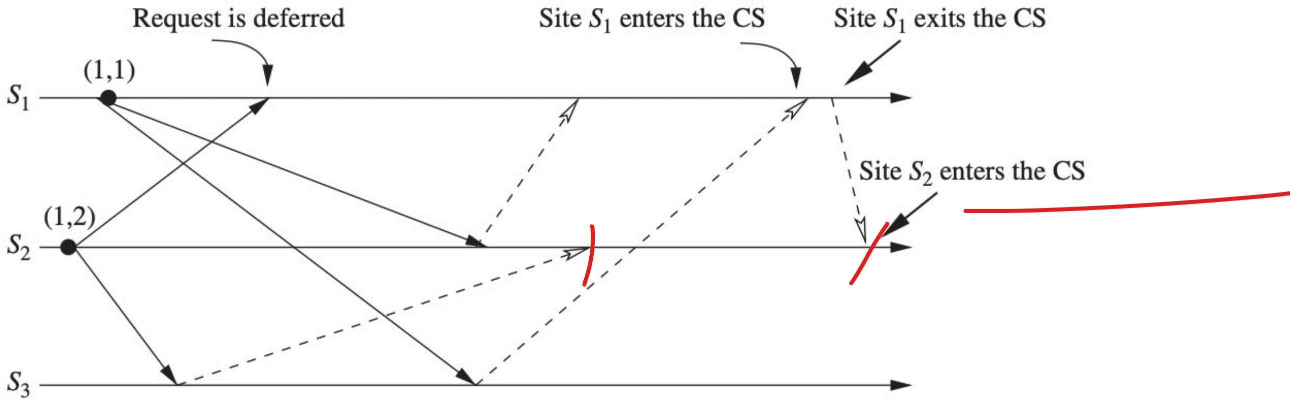


Figure 9.10 Site S_2 enters the CS.




thank you


LIKE
COMMENT
SHARE
SUBSCRIBE

(X)

P_1 P_2 seq-1 seq-2

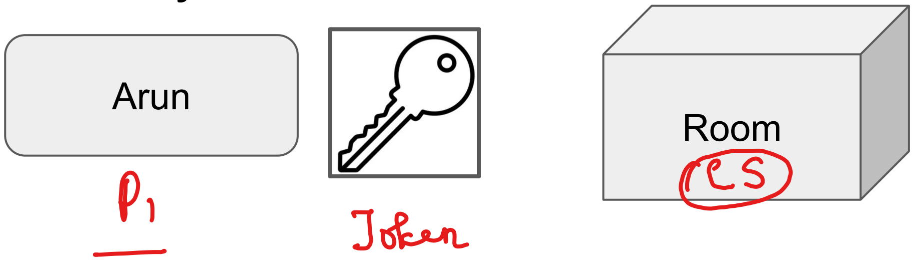
P_3 (X)

Token Based Algorithm

1. a unique token is shared among the sites.
2. A site is allowed to enter its CS if it possesses the token.
3. A site holding the token can enter its CS repeatedly until it sends the token to some other site.
4. First, token-based algorithms use sequence numbers instead of timestamps.
   a. Every request for the token contains a sequence number and the sequence numbers of sites advance independently.
   b. A site increments its sequence number counter every time it makes a request for the token.
5. algorithm guarantees mutual exclusion because site holds the token during the execution of the CS..

Suzuki–Kasami's broadcast algorithm

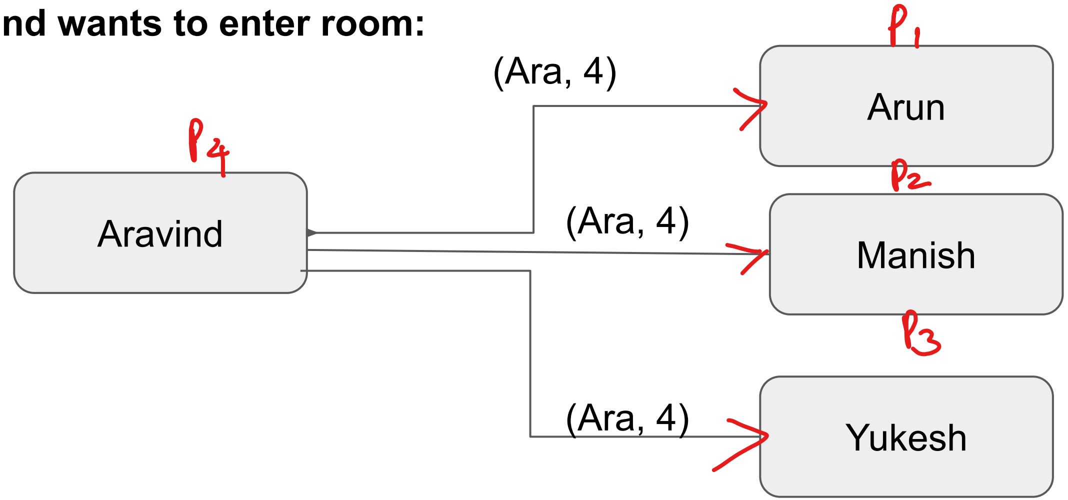
Arun is already in Room:



work @ room over

token/key is sent out

Aravind wants to enter room:



1 REQUEST

(sender-id, seq-no)

Problem 1 : Outdated vs Recent Request

P_1 P_2, P_3

@Arun

P_4

1. (Aravind,4)
2. (Aravind,20)
3. (Aravind,4)

1-2-3- . . . . . -20

@ every process

RN ⇒ last req seq. no.

RN_{P_1} [- - - 0] (P_1 P_2 P_3 P_4)

@ P_1

1. $(P_4, 4) \Rightarrow \underset{max}{\underline{0, 4}} \Rightarrow RN_{P_1} [0\ 0\ 0\ 4]$

2. $(P_4, 20) \Rightarrow \underset{max}{\underline{4, 20}} \Rightarrow RN_{P_1} [0\ 0\ 0\ 20]$

3. $(P_4, 4) \Rightarrow 20, 4 \Rightarrow$ Outdated

Problem 2: How to inform I have completed my work in CS?

P_1 — P_2

req 1pm   8pm

P_1 P_2 P_3

RN_3 [4 2 -]

Process	CS completed seq no	Request Seq no
P1	3	4
P2	1	2
P3	4	-

LN: Stores CS completed seq no

LN [3 1 4]

P_1

3+1 ⇒ 4

LN + 1 ⇒ RN ?

queue add

Requests at P3,

1 (P1,4)

2 (P2,2)

3 (P1,2)

1+1 ⇒ 2 ✓

-

outdated

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 $RN_j[i] = LN[i] + 1$.

$S_i \rightarrow S_j$

RN_i RN_j

Executing the critical section:

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

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

$S_3 \Rightarrow 10$

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

$LN[3] = 10$

(e) For every site S_j whose i.d. is not in the token queue, it appends its i.d. 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 i.d. from the token queue and sends the token to the site indicated by the i.d.

Algorithm 9.7 Suzuki–Kasami's broadcast algorithm.

Problem Explanation

1. **How to distinguishing an outdated REQUEST message from a current REQUEST message**
    a. Due to variable message delays, a site may receive a token request message after the corresponding request has been satisfied.
    b. If a site cannot 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, but it may seriously degrade the performance by wasting messages and increasing the delay at sites that are genuinely requesting the token.
    c. Therefore, appropriate mechanisms should implemented to determine if a token request message is outdated.
2. **How to determine which site has an outstanding request for the CS?**
    a. 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.
    b. The problem is complicated because when a site Si receives a token request message from a site Sj, site Sj may have an outstanding request for the CS.
    c. However, after the corresponding request for the CS has been satisfied at Sj, an issue is how to inform site Si (and all other sites) efficiently about it.

Problem Solution

1

Outdated REQUEST messages are distinguished from current REQUEST messages in the following manner: a REQUEST message of site S_j has the form REQUEST(j, n) where n ($n = 1, 2, \ldots$) is a sequence number that indicates that site S_j is requesting its nth CS execution. A site S_i keeps an array of integers $RN_i[1, \ldots, N]$ 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 $RN_i[j] := max(RN_i[j], n)$. Thus, when a site S_i receives a REQUEST(j, n) message, the request is outdated if $RN_i[j] > n$.

2

Sites with outstanding requests for the CS are determined in the following manner: the token consists of a queue of requesting sites, Q, and an array of integers $LN[1, \ldots, N]$, 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. Token array $LN[1, \ldots, N]$ permits a site to determine if a site has an outstanding request for the CS. Note that at site S_i if $RN_i[j] = LN[j]+1$, then site S_j is currently requesting a token. After executing the CS, a site checks this condition for all the j's to determine all the sites that are requesting the token and places their i.d.'s in queue Q if these i.d.'s are not already present in Q. Finally, the site sends the token to the site whose i.d. is at the head of Q.

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 9.3 *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. $\square$


thank you


LIKE
COMMENT
SHARE
SUBSCRIBE