Outline for February 8, 2001

1. Greetings and felicitations!
2. Maekawa's's
   a. Request sets satisfy the following conditions:
      i. for all $1 \le i, j \le n$ with $i \ne j$, $R_i \cap R_j \ne \varnothing$
      ii. for all $1 \le i \le n$, $p_i \in R_i$
      iii. for all $1 \le i \le n$, $| R_i | = K$
      iv. p_j is contained in K number of R_i.
   b. Idea: every pair of processes has has a mediator (in both request sets) to mediate conflicts
   c. Only one outstanding REPLY per process, so each process gives permission to enter to only one other process
   d. Assumes messages delivered in order which they are sent
   e. To avoid deadlock, can INQUIRE whether someone cannot get it
   f. A FAILED message just means someone else is going in out of order
   g. Performance: between $3\sqrt{N}$ and $5\sqrt{N}$ messages
3. Sanders' generalized protocol
   a. Inform set I_i is set of processes to be informed when p_i enters or leaves CS
   b. Status set ST_i contains pids for which p_i maintains status information; note: $p_i \in I_j \Rightarrow p_j \in ST_i$
   c. If $p_i \in I_i$ for all i, then necessary and sufficient conditions to guarantee mutual exclusion are:
      i. $I_i \subseteq R_i$
      ii. $I_i \cap I_j \ne \varnothing$ or ($p_i \in$ Rj and $p_j \in R_i$)
4. Suzuki-Kasami's broadcast protocol
   a. token-based
   b. uses sequence numbers, not clocks
   c. token has sequence numbers, associated queue
   d. how to handle stale requests? token's sequence number too high
5. Raymond's tree-based protocol
   a. token-based
   b. think of token as at root of tree, root moves around

Maekawa's Distributed Mutual Exclusion Protocol

Introduction

Maekawa's algorithm uses sets S_i of fewer than n processes. Also, a process p_i locks all the nodes in S_i by having only one reply message out at a time. Each process has a queue of unsatisfied requests ordered by timestamp.

Notation

- n processes $p_1, \ldots, p_n$
- t_j timestamp

Protocol

1. To enter the critical section, p_i sends REQUEST(t_i, i) to all sites in S_i.
2. When p_j receives a REQUEST(t_i, i) message:
   a. if it has not sent a REPLYmessage to any site since the last RELEASE message p_j received, p_j sends REPLY(t_j, ,j) to p_i.
   b. Otherwise, there is a process p_k such that p_j has already sent a REPLY(t_j, ,j) message to p_k. Then:
      i. if (t_i, i) < (t_k, k), p_j sends an INQUIRE(j) message to p_k and places the REQUEST on its queue.
      ii. Otherwise, p_j sends a FAILED(j) message to p_i.
3. When p_k receives an INQUIRE(j) message, it sends a YEILD(k) message if either of the following conditions are met:
   a. p_k has received a FAILED message from a site in S_k
   b. p_k has sent a YIELD(m) message to a site $p_m \in S_k$ and has not received a REPLY in reply.

   Otherwise p_k does nothing.
4. When p_j receives a YIELD(k) message, p_j places REQUEST(t_k, k) in the queue and sends a REPLY(t_j, j) to the site whose request is first in the queue.
5. When p_j receives a RELEASE(t_i, i) message, it sends a REPLY(t_j, ,j) message to the next site in the queue and deletes the entry from the queue. If the queue is empty, p_j updates its state to reflect that no REPLY message has been sent to any site since the last RELEASE message p_j received.
6. p_i enters the critical section when it has received REPLY messages from all processes in S_i.
7. When p_i leaves the critical section, it sends RELEASE(t_i, i) to all sites in S_i.

Example

There are 13 processes. Initially, all logical clocks are set to 0. The sets are:

S_1 = { 1, 2, 3, 4 } S_4 = { 4, 6, 10, 11 } S_7 = { 2, 7, 10, 13 } S_{10} = { 3, 5, 10, 12 }
S_2 = { 2, 5, 8, 11 } S_5 = { 1, 5, 6, 7 } S_8 = { 1, 8, 9, 10 } S_{11} = { 1, 11, 12, 13 }
S_3 = { 3, 6, 8, 13 } S_6 = { 2, 6, 9, 12 } S_9 = { 3, 7, 9, 11 } S_{12} = { 4, 7, 8, 12 }
S_{13} = { 4, 5, 9, 13 }

p11 sends REQUEST(0, 11) to p1, p12, p13.
p12 receives REQUEST(0, 11) and sends REPLY(1, 12) to p11.
p13 receives REQUEST(0, 11) and sends REPLY(1, 13) to p11.
p7 sends REQUEST(0, 7) to p2, p10, p13.
p2 receives REQUEST(0, 7) and sends REPLY(1, 2) to p7.
p10 receives REQUEST(0, 7) and sends REPLY(1, 10) to p7.
p8 sends REQUEST(0, 8) to p1, p9, p10.
p1 receives REQUEST(0, 8) and sends REPLY(1, 1) to p8.
p9 receives REQUEST(0, 8) and sends REPLY(1, 9) to p8.
p10 receives REQUEST(0, 8). It already sent a REPLY to p7. p7's request is timestamped (0, 7) < (0, 8). p10 sends FAILED(10) to p8.
p1 receives REQUEST(0, 11). It already sent a REPLY to p8. p8's request is timestamped (0, 8) < (0, 11). p1 sends FAILED(1) to p11.
p13 receives REQUEST(0, 7). It already sent a REPLY to p11. p11's request is timestamped (0, 11), and (0, 7) < (0, 11). p13 sends INQUIRE(13) to p11.
p11 receives INQUIRE(13). It has received a FAILED(1) message from p1. p11 sends YIELD(11) to p13.
p13 receives YIELD(11). It now sends REPLY(2, 7) to p7 and places REQUEST(0, 11) in its queue.
p7 has received replies from all processes in S_7. It enters the critical section.
p7 leaves the critical section and sends RELEASE(1000, 7) to p2, p10, p13.
p10 receives RELEASE(1000, 7).
p10 sends REPLY(1001, 8).
p8 has received replies from all processes in S_8. It enters the critical section.
p8 leaves the critical section and sends RELEASE(1003, 8) to p1, p9, p10.
p1 receives RELEASE(1003, 8).
p1 sends REPLY(1004, 1).
p11 has received replies from all processes in S_{11}. It enters the critical section.
p11 leaves the critical section and sends RELEASE(1005, 11) to p1, p12, p13.

Sanders' Generalized Protocol

Introduction

This protocol is a generalization of the previous protocols.

Notation

- n processes
- p_i process
- R_i request set for p_i
- I_i inform set for p_i
- ST_i status set for p_i
- *CSSTAT* site's knowledge of state of critical section

Protocol

1. To request entry, p_i sends REQUEST(t_i, i) to all sites in R_i.
2. When a site p_i gets a REQUEST(t_j, j) message:
   a. it places the request onto its queue, which is ordered by timestamps
   b. if *CSSTAT* says the critical section is free, p_i sends a GRANT message to the first process p_f in the queue and deletes its entry from the queue. If $p_f \in ST_i$, then *CSSTAT* is set to indicate that p_f is in the critical section.
3. When p_i has received GRANT messages from all processes in R_i, p_i enters the critical section.
4. When p_i leaves the critical section, p_i sends a RELEASE message to every site in I_i.
5. When p_i receives a RELEASE message:
   a. *CSSTAT* is set to free
   b. If p_i queue is not empty, p_i sends a GRANT to the first process p_f in the queue and deletes its entry from the queue. If $p_f \in ST_i$, then *CSSTAT* is set to indicate that p_f is in the critical section.
   c. Repeat step b until either *CSSTAT* indicates a process has entered the critical section, or p_f's queue is empty.

Example

There are three processes, p_1, p_2, and p_3. p_1 and p_3 seek mutually exclusive access to a shared resource. Let:
$I_1 = \{ p_1, p_3 \}$, $I_2 = \{ p_2 \}$, $I_3 = \{ p_3 \}$, so $ST_1 = \{ p_1 \}$, $ST_2 = \{ p_2 \}$, $ST_3 = \{ p_1, p_3 \}$; and
$R_1 = \{ p_1, p_2, p_3 \}$, $R_2 = \{ p_1, p_2, p_3 \}$, $R_3 = \{ p_2, p_3 \}$
These satisfy the criteria that, for all i, $p_i \in I_i$, $I_i \subseteq R_i$, and for all pairs (i, j), either $I_i \cap I_j \neq \varnothing$ or ($p_i \in$ Rj and $p_j \in R_i$).
Initially:

p1 state: C1 = 0, Q1 empty, CSSTAT1 empty
p2 state: C2 = 0, Q2 empty, CSSTAT2 empty
p3 state: C3 = 0, Q3 empty, CSSTAT3 empty

p1 sends Q(0,1) to p1, p2 and p3; p1's state now C1 = 1, Q1 empty, CSSTAT1 empty, GRANTS1 empty
p1 receives Q(0,1) from p1; p1's state now C1 = 1, Q1 = Q(0,1), CSSTAT1 empty, GRANTS1 empty
p1 sends G(1,1) to p1; p1's state now C1 = 2, Q1 empty, CSSTAT1 = p1, GRANTS1 empty
p1 receives G(1,1) to p1; p1's state now C1 = 2, Q1 empty, CSSTAT1 = p1, GRANTS1 G(1,1)
p3 sends Q(0,3) to p2 and p3; p3's state now C3 = 1, Q3 empty, CSSTAT3 empty, GRANTS3 empty
p3 receives Q(0,3) from p3; p3's state now C3 = 1, Q3 = Q(0,3), CSSTAT3 empty, GRANTS3 empty
p3 sends G(1,3) to p3; p3's state now C3 = 2, Q3 empty, CSSTAT3 = p3, GRANTS3 empty
p3 receives G(1,3) from p3; p3's state now C3 = 2, Q3 empty, CSSTAT3 = p3, GRANTS3 G(1,3)
p2 receives Q(0,1) from p1; p2's state now C2 = 2, Q2 = Q(0,1), CSSTAT2 empty, GRANTS2 empty
p2 sends G(2,2) to p1; p2's state now C2 = 3, Q2 empty, CSSTAT2 empty, GRANTS2 empty
p2 receives Q(0,3) from p3; p2's state now C2 = 3, Q2 = Q(0,3), CSSTAT2 empty, GRANTS2 empty
p2 sends G(3,2) to p3; p2's state now C2 = 4, Q2 empty, CSSTAT2 empty, GRANTS2 empty
p3 receives Q(0,1) from p1; p3's state now C3 = 2, Q3 = Q(0,1), CSSTAT3 = p3, GRANTS3 G(1,3)
p1 receives G(2,2) from p2; p1's state now C1 = 2, Q1 empty, CSSTAT1 = p1, GRANTS1 G(1,1), G(2,2)
p3 receives G(3,2) from p2; p3's state now C3 = 3, Q3 = Q(0,1), CSSTAT3 = p3, GRANTS3 G(1,3), G(3,2)
p3 enters its critical section
p3 exits its critical section
p3 sends R(3,3) to p3; p3's state now C3 = 4, Q3 empty, CSSTAT3 = p3, GRANTS3 empty
p3 receives R(3, 3) from p3; p3's state now C3 = 4, Q3 = Q(0,1), CSSTAT3 empty, GRANTS3 empty
p3 sends G(4, 3) to p1; p3's state now C3 = 5, Q3 empty, CSSTAT3 = p1, GRANTS3 empty
p1 receives G(4,3) from p3; p1's state now C1 = 4, Q1 empty, CSSTAT1 = p1, GRANTS1 G(1,1), G(2,2), G(4,3)
p1 enters its critical section
p1 exits its critical section
p1 sends R(4, 1) to p1, p3; p1's state now C1 = 5, Q1 empty, CSSTAT1 = p1, GRANTS1 empty
p1 receives R(4,1) from p1; p1's state now C1 = 5, Q1 empty, CSSTAT1 empty, GRANTS1 empty
p3 receives R(4,1) from p1; p3's state now C3 = 5, Q3 empty, CSSTAT3 empty, GRANTS3 empty

Suzuki-Kasami Broadcast Protocol

Introduction

This is a token-based protocol. Unlike non-token-based ones, it uses the token's being possessed by a site to provide ordering of requests. Clocks and virtual time do not play a role; but order of arrival does.

Notation

- n processes
- p_i process
- $R_i[j]$ largest sequence number p_i has received in a REQUEST message from p_j
- $L[i]$ sequence number of request that p_i has most recently executed
- Q queue (sequence) of sites requesting token
- T = (Q, L) token

Protocol

1. To request entry, if p_i does not have the token, it increments its sequence number $R_i[i]$ and then sends REQUEST(i, s), $s = R_i[i]$, to all other sites.
2. When p_i receives REQUEST(i, s) from p_j, p_i sets $R_i[j] = \max(R_i[j], s)$. If p_i has the token and $R_i[j] = L[j] + 1$, it sends the token to p_j.
3. If p_i is requesting entry and it has or receives the token, it enters the critical section.
4. When p_i finishes executing the critical section:
   a. it sets $L[i] = R_i[i]$;
   b. for every j not in Q and for which $R_i[j] = L[j] + 1$, p_i appends j to Q; and
   c. if Q is not empty, p_i deletes the first element f of Q and sends the token to p_f.

Example

There are three processes, p_1, p_2, and p_3. p_1 and p_3 seek mutually exclusive access to a shared resource.
Initially:the token is at p2 and the token's state is L = [0, 0, 0] and Q empty;
p1's state is C1 = 0, R1 = [0, 0, 0]; p3's state is C1 = 0, R2 = [0, 0, 0]; and p3's state is C3 = 0, R3 = [0, 0, 0]
p1 sends R(1, 1) to p2 and p3; p1's state is C1 = 1, R1 = [1, 0, 0]
p3 sends R(3, 1) to p1 and p2; p3's state is C3 = 1, R3 = [0, 0, 1]
p2 receives R(1, 1) from p1; p2's state is C2 = 1, R2 = [1, 0, 0], holding token
p2 sends the token to p1
p1 receives R(3, 1) from p3; p1's state is C1 = 1, R1 = [1, 0, 1]
p3 receives R(1, 1) from p1; p3's state is C3 = 1, R3 = [1, 0, 1]
p1 receives the token from p2
p1 enters the critical section
p1 exits the critical section and sets the token's state to L = [1, 0, 0] and Q = (3)
p1sends the token to p3; p1's state is C1 = 2, R1 = [1, 0, 1], holding token, token's state is L = [1, 0, 0] and Q empty
p3 receives the token from p1; p3's state is C3 = 1, R3 = [1, 0, 1], holding token
p3 enters the critical section
p3 exits the critical section and sets the token's state to L = [1, 0, 1] and Q empty

Raymond's Tree-Based Protocol

Introduction

This is a token-based protocol. The nodes are arranged in a binary tree, and one acquires the token by going up the tree. The token is always kept at the root, so the tree needs to rearrange itself as the token floats from site to site.

Notation

- n processes
- p_i process
- Q_i request queue (sequence) of sites associated with process p_i
- H_i holder variable associated with process p_i
- T token

Protocol

1. To request entry, if p_i does not have the token, it sends a REQUEST(i) message to the node named in H_i unless Q_i is not empty (because then it has already sent a REQUEST(i) but has not yet received the token). It adds the request to Q_i.
2. When p_i receives REQUEST(j) from p_j:
   a. if p_i does not have the token, it places the REQUEST(j) on Q_i and sends a REQUEST(i) message along (provided that it is not waiting for a response to an earlier REQUEST(i).
   b. if p_i has the token, it sends the token to p_j and sets H_i to j.
3. If p_i is requesting entry and receives the token:
   a. if i is not the first entry in Q_i, it deletes the first entry j from Q_i and forwards the token to p_j. It then sets H_i to j. If Q_i is not empty, p_i sends REQUEST(i) to p_j.
   b. if i is the first entry in Q_i, p_i deletes i from Q_i and enters the critical section.
4. When p_i finishes executing the critical section:
   a. if Q_i is not empty, it deletes the first entry j from Q_i, sends the token to p_j, and sets H_i to j
   b. if after step a Q_i is not empty, p_i sends REQUEST(i) to p_j.

Example

There are six processes, p_1 through p_6. p_1 and p_5 seek mutually exclusive access to a shared resource, and later p_3 will request it.

Initially: p4 has the token;

p1's state is C1 = 0, HOLDER2 = p3, Q1 empty
p2's state is C2 = 0, HOLDER2 = p3, Q2 empty
p3's state is C3 = 0, HOLDER3 = p4, Q3 empty
p4's state is C4 = 0, HOLDER4 = p4, Q4 empty, holding token
p5's state is C5 = 0, HOLDER5 = p4, Q5 empty
p6's state is C6 = 0, HOLDER6 = p5, Q6 empty

p1 sends Q(1) to p3; p1's state is C1 = 1, HOLDER2 = p3, Q1 = Q(1).
p5 sends Q(5) to p4; p5's state is C5 = 1, HOLDER5 = p4, Q5 = Q(5).
p3 receives Q(1) from p1; p3's state is C3 = 0, HOLDER3 = p4, Q3 empty.
p3 sends Q(3) to p4; p3's state is C3 = 1, HOLDER3 = p4, Q3 = Q(1).
p4 receives Q(5) from p5; p4's state is C4 = 0, HOLDER4 = p4, Q4 empty, holding token.
p4 sends token to p5; p4's state is C4 = 1, HOLDER4 = p5, Q4 empty.
p4 receives Q(3) from p3; p4's state is C4 = 1, HOLDER4 = p5, Q4 empty.
p4 sends Q(4) to p5; p4's state is C4 = 2, HOLDER4 = p5, Q4 = Q(3).
p5 receives token from p4; p5's state is C5 = 1, HOLDER5 = p4, Q5 = Q(5).
p5 resets state to C5 = 1, HOLDER5 = p4, Q5 empty, holding token.
p5 enters the critical section
p5 leaves the critical section
p5 receives Q(4) from p4; p5's state is C5 = 1, HOLDER5 = p4, Q5 empty, holding token.
p5 sends token to p4; p5's state is C5 = 2, HOLDER5 = p4, Q5 empty.
p3 sends Q(3) to p4; p3's state is C3 = 2, HOLDER3 = p4, Q3 = Q(1) Q(3).
p4 receives Q(3) from p3; p4's state is C4 = 2, HOLDER4 = p5, Q4 = Q(3).
p4's state is C4 = 3, HOLDER4 = p5, Q4 = Q(3)Q(5) [it sends nothing as it is waiting for a response]
p4 receives token from p5; p4's state is C4 = 3, HOLDER4 = p5, Q4 = Q(3) Q(5), holding token.
p4 sends token to p3; p4's state is C4 = 3, HOLDER4 = p3, Q4 = Q(5).
p4 sends Q(4) to p3; p4's state is C4 = 3, HOLDER4 = p3, Q4 = Q(5).
p3 receives token from p4; p3's state is C3 = 2, HOLDER3 = p4, Q3 = Q(1) Q(3), holding token.
p3 sends token to p1; p3's state is C3 = 3, HOLDER3 = p1, Q3 = Q(3).
p3 sends Q(3) to p1; p3's state is C3 = 4, HOLDER3 = p1, Q3 = Q(3).
p1 receives token from p3; p1's state is C1 = 1, HOLDER1 = p3, Q1 = Q(1), holding token.
p1 resets state to C1 = 1, HOLDER1 = p3, Q1 empty, holding token.
p1 enters the critical section
p1 leaves the critical section
p1 receives Q(3) from p3; p1's state is C1 = 1, HOLDER1 = p3, Q1 empty, holding token.
p1 sends token to p3; p1's state is C1 = 2, HOLDER1 = p3, Q1 empty.
p3 receives token from p1; p3's state is C3 = 4, HOLDER3 = p1, Q3 = Q(3), holding token.
p3 receives Q(4) from p4; p3's state is C3 = 4, HOLDER3 = p1, Q3 = Q(3) Q(4).
p3 resets state to C3 = 4, HOLDER3 = p1, Q3 = Q(4).
p3 enters the critical section
p3 leaves the critical section
p3 sends token to p4; p3's state is C3 = 5, HOLDER3 = p4, Q3 empty, holding token.
p4 receives token from p3; p4's state is p4's state is C4 = 3, HOLDER4 = p3, Q4 = Q(5).
p4 sends token to p5; p4's state is C4 = 4, HOLDER4 = p5, Q4 empty.
p5 receives token from p4; p5's state is C5 = 2, HOLDER5 = p4, Q5 empty.