MAEKAWA'S DISTRIBUTED MUTEX ALGORITHM

Reference: 1. Mukesh Singhal & N.G. Shivaratri, Advanced Concepts in Operating Systems,

2. George Coulouris, Jean Dollimore and Tim Kindberg, "Distributed Systems Concepts and Design", Fifth Edition, Pearson Education, 2012

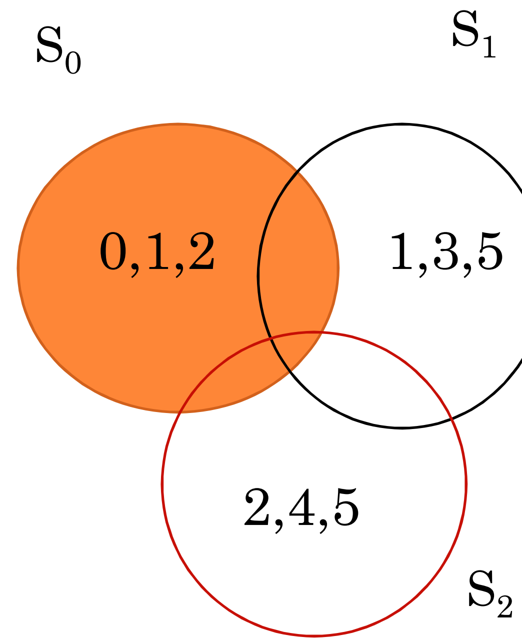
Maekawa's algorithm

- With each process i, associate a subset S_i. Divide the set of processes into subsets that satisfy the following two conditions:

  $i \in S_i$

  $\forall i,j : 0 \leq i,j \leq n-1 \mid S_i \cap S_j \neq \emptyset$

- **Main idea**. Each process i is required to receive permission from S_i only. Correctness requires that multiple processes will never receive permission from all members of their respective subsets.



Maekawa's algorithm

Example. Let there be seven processes 0, 1, 2, 3, 4, 5, 6

$S_0 = \{0, 1, 2\}$
$S_1 = \{1, 3, 5\}$
$S_2 = \{2, 4, 5\}$
$S_3 = \{0, 3, 4\}$
$S_4 = \{1, 4, 6\}$
$S_5 = \{0, 5, 6\}$
$S_6 = \{2, 3, 6\}$

MAEKAWA'S ALGORITHM

Version 1 {Life of process I}

1. Send timestamped ***request*** to each process in $\mathbf{S_i}$.
2. Request received → send ***ack*** to process with the *lowest timestamp*. Thereafter, "**lock**" (i.e. **commit**) yourself to that process, and keep others waiting.
3. Enter CS if you receive an *ack* from **each member** in S_i.
4. To exit CS, send *release* to every process in S_i.
5. Release received → **unlock** yourself. Then send ack to the next process with the lowest timestamp.

S_0= {0, 1, 2}

S_1= {1, 3, 5}

S_2= {2, 4, 5}

S_3= {0, 3, 4}

S_4= {1, 4, 6}

S_5= {0, 5, 6}

S_6= {2, 3, 6}

MAEKAWA'S ALGORITHM-VERSION 1

ME1. ***At most one process can enter its critical section at any time.***

Let **i** and **j** attempt to enter their Critical Sections

$S_i \cap S_j \neq \emptyset$ implies there is a process $\mathbf{k} \in S_i \cap S_j$

Process **k** will **never** send ack to both.

So it will act as the arbitrator and establishes ME1

$S_0 = \{0, 1, 2\}$

$S_1 = \{1, 3, 5\}$

$S_2 = \{2, 4, 5\}$

$S_3 = \{0, 3, 4\}$

$S_4 = \{1, 4, 6\}$

$S_5 = \{0, 5, 6\}$

$S_6 = \{2, 3, 6\}$

MAEKAWA'S ALGORITHM-VERSION 1

ME2. No deadlock. ***Unfortunately deadlock is possible!*** ***Assume 0, 1, 2 want to enter their critical sections.***

From S_0= **{0,1,2}**, **0,2** send *ack* to **0**, but **1** sends *ack* to **1**;

From S_1= **{1,3,5}**, **1,3** send *ack* to **1**, but **5** sends *ack* to **2**;

From S_2= **{2,4,5}**, **4,5** send *ack* to **2**, but **2** sends *ack* to **0**;

Now, 0 waits for 1 **(to send a release),** 1 waits for 2 **(to send a release),** , and 2 waits for 0 **(to send a release),** . So deadlock **is** possible!

S_0=	{0, 1, 2}
S_1=	{1, 3, 5}
S_2=	{2, 4, 5}
S_3=	{0, 3, 4}
S_4=	{1, 4, 6}
S_5=	{0, 5, 6}
S_6=	{2, 3, 6}

Maekawa's algorithm-Version 2

Avoiding deadlock

If processes always receive messages **in increasing order of timestamp**, then deadlock "could be" avoided. But this is too strong an assumption.

Version 2 uses three ***additional*** messages:

- ***failed***
- ***inquire***
- ***relinquish***

$S_0 = \{0, 1, 2\}$

$S_1 = \{1, 3, 5\}$

$S_2 = \{2, 4, 5\}$

$S_3 = \{0, 3, 4\}$

$S_4 = \{1, 4, 6\}$

$S_5 = \{0, 5, 6\}$

$S_6 = \{2, 3, 6\}$

Maekawa's algorithm-Version 2

New features in version 2

- Send ***ack*** and set lock as usual.
- If lock is set and a request with a larger timestamp arrives, send ***failed*** *(you have no chance)*. If the incoming request has a lower timestamp, then send ***inquire*** *(are you in CS?) to the locked process*.
- Receive ***inquire*** and at least one ***failed*** message → send ***relinquish***. The recipient resets the lock.

$S_0 = \{0, 1, 2\}$

$S_1 = \{1, 3, 5\}$

$S_2 = \{2, 4, 5\}$

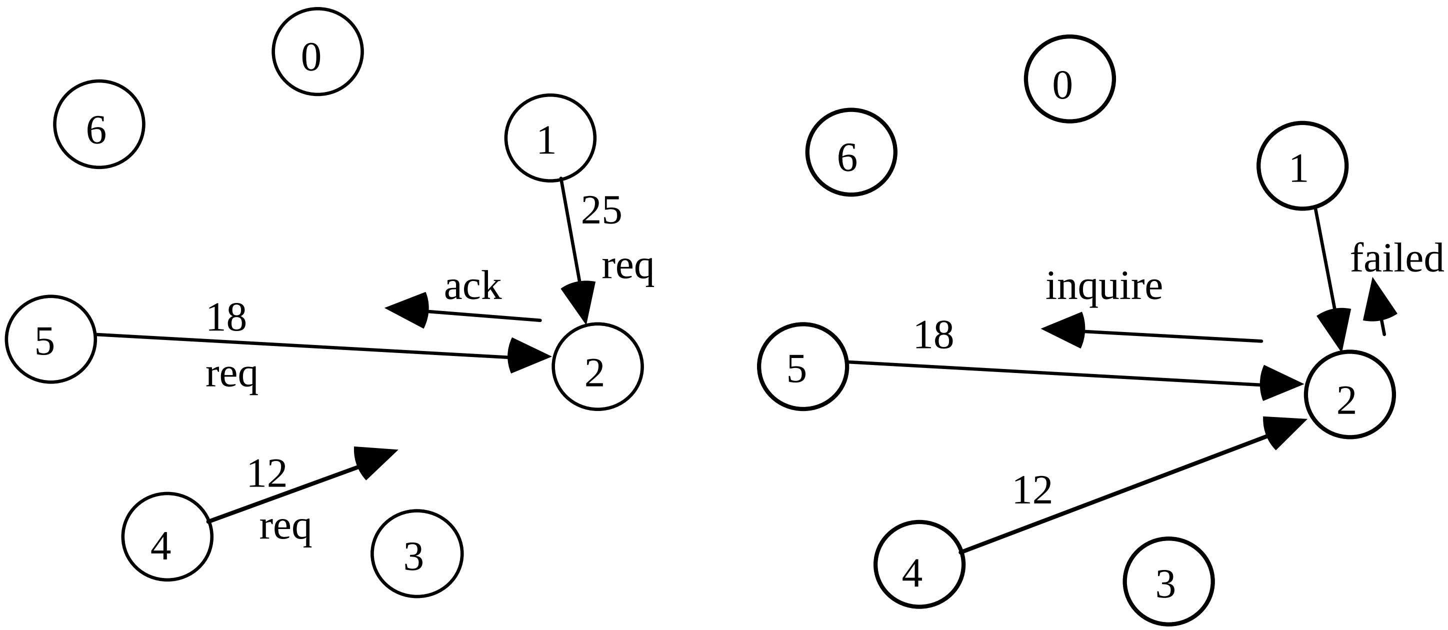
$S_3 = \{0, 3, 4\}$

$S_4 = \{1, 4, 6\}$

$S_5 = \{0, 5, 6\}$

$S_6 = \{2, 3, 6\}$

Maekawa's algorithm-Version 2



COMMENTS

- Let $K = |S_i|$. Let each process be a member of D subsets. When N = 7, K = D = 3. When K=D, N = K(K-1)+1. So $K = O(\sqrt{N})$

- The message complexity of Version 1 is $3\sqrt{N}$. Maekawa's analysis of Version 2 reveals a complexity of $7\sqrt{N}$

- *Sanders identified a bug in version 2* …