# The Prioritized and Distributed Synchronization in Distributed Groups

Michel Trehel and Ahmed Housni

Université de Franche Comté, 16, route de Gray 25000 Besançon France,
{trehel, housni} @lifc.univ-fcomte.fr

**Abstract.** A simple and cheap algorithm is presented to allow prioritized mutual exclusion. There are several groups. All the members of a same group have the same level priority. Our algorithm is a token-based algorithm. Each group of participants (*site*) is represented by a tree structure. Inside a group, the requests are recorded in a global queue which circulates simultaneously and together with the token. The participant holding the token is the root of the tree linked to the router of the group. When a router transmits the token to another group, it preserves the role of the router in its group. The relation between routers is also represented by an rooted tree, the root is the router of the group holding the token. Besides a static logical structure similar to that used in Raymond's algorithm, our algorithm manages a global requester queue. If the requesting site and the owner of the token are in the same group, there is a reorganization of the group tree. If they are not in the same group, the tree of the requesting site and the tree of routers are reorganized. Algorithm in one group and extension to some groups are presented.

## 1 Introduction

Prioritized mutual exclusion can be applied to speech allocation in a multi-role conference. The speakers of each role constitute a group and there is a priority level by group. At a given moment, only one participant of only one group may speak to the other ones. When the situation is limited to two groups, prioritized mutual exclusion corresponds, for example, to a group of trainers and a group of trainees, in teleteaching. The resource may be the speech. Trainers have priority on trainees. Some works have been presented concerning the prioritized mutual exclusion. A. Goscinski's [1], algorithms lean on request broadcasts. The average number of messages is O(*n)* where n is the number of sites. K. Harathi and T. Johnson [2] propose prioritized spin lock mutual exclusion algorithms. The blocked processes spin on locally stored or cached variables. The performances are improved, compared with the previous ones. Nevertheless, it is O(n). T. Johnson and R. Newman-Wolfe [4] developed three algorithms giving approximately the same performances. They aimed at synchronizing the access to processor memory. One of them uses a rooted tree as in Raymond's approach [8].

The other two algorithms use a path compression technique to reduce the number of messages. B. M. K. Qazzaz [5] gave an algorithm based on a binary tree. The prioritized nodes have a special position in the tree, what gives strong constraints to the system. Mueller [7] proposed an idea derived from the M. Naimi and M. Trehel [9] algorithm, associating a priority level with the requests. For that, the queue distributed in [9] is replaced in [7] by local queues. Every site owns a local queue. When it receives a request, it compares the priority of the queue's sites with the arriving request priority, to reorder the queue. The average number of messages is O(Log(n)). In our algorithm, there is a priority level by group. The presentation of the algorithm is modular: one group (all the same priority), and many groups (one priority level by group).

## 2 Our Algorithm for One Group

### 2.1 General Presentation

In this paper, the term "tree" is used when there is neither orientation neither root, and "rooted tree", when there is an orientation and a specific root. Both terms are included, because both concepts are needed. Our algorithm in one group is closed to K. Raymond's algorithm [8]. Let us describe the idea. The participants are structured as a logical tree. We will speak about the choice of the tree, function of performance considerations. A root is chosen in the initial situation. The root is said privileged because it owns the token. The choice of the root transforms the tree in a rooted tree. Every site owns a local variable called its "father", which indicates the direction of the root. "Father" is nil at the root. Look at the figure 1 as an example. Suppose the root A is in the critical section. If the nonprivileged node D wishes to enter the critical section, it sends a "request" message to its "father" C. When receiving the "request" message, the node C, if it is not the root, forwards the message to its "father" B. Thus a series of "request" messages travels along the path from the requesting node D to the root A. The message "request" is put in the memory of the root.

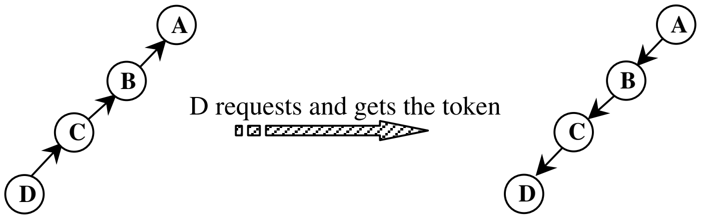


**Fig. 1.** Transformation

When the root releases the critical section, it sends the token to its neighbor B, in the direction of the requesting node. If A has more than a neighbor, a specific technique is necessary to determine to which node A must send the token. While sending the token, A looses its quality of root and its "father" becomes B. Now, the "father" of B is nil.

There is no change of the underlying tree, nevertheless the rooted tree is changed and the new root is B. If B is not the requesting node (B is not D), B forwards the token to its neighbor, in the direction of D. Thus a series of "token" messages travels from A to D. When it is arrived in D, D owns the token and enters the critical section. The rooted tree is reorganized and the new root is D.

## 2.2 Hypotheses

### 2.2.1. Network

The network is the initial rooted tree. The link between two sites is bi-directional, so that the communications can be sent after a reorganization of the rooted tree.

### 2.2.2 Communication

There is no loss, duplication or modification of messages. Transmission times are random, but finite. If several messages arrived simultaneously in a site, they are treated sequentially. Nevertheless a message can arrive when a site is in the critical section. From one site to another, the messages arrive in the order in which they have been sent. Communications run at least two times faster than the passage in critical section.

### 2.2.3 Critical Section

A site must wait reception of the critical section and releasing it before requesting it again. It stays in the critical section for a finite time.

### 2.2.4 Specifications

Mutual exclusion: Every site which requests the critical section has to obtain it after a finite time. However, it is not true when there are groups with different priorities.

## 2.3 Principles of the Algorithm

### 2.3.1 Token

It is a token-based algorithm. When the root releases the critical section, it sends the token in the direction of the requesting site. Every site crossed during the transmission of the token earns then loses the quality of the root.

### 2.3.2 Routing

The principle of the transmission of the request to the root was seen in the general presentation: the variable "father" gives the direction of the root. The token has to come back from the root to the requesting site. For that, the following technique is used: Every site i owns a routing table, i. e., it knows for every site j the first node on the path from i to j in the tree. This principle is used in Internet (RIP). These tables remain unchanged. The following is an example of the routing table of A (table 1 and Figure 2).

**Table 1.** Routing table of A

| *A* | *B* | *C* | *D* | *E* | *F* | *G* |
|---|---|---|---|---|---|---|
| nil | B | C | C | C | C | C |

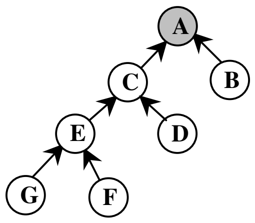


**Fig. 2.** the direction of the root

### 2.3.3 Queue

The requests are queued in the queue located at the root. When the root releases the critical section, it serves the head of the queue (FIFO service).

## 3 Example

***E requests the critical section:***

The variable "request" is true for E (Table 3). As E is not the root, it transmits the request to its father.

**Table 2.** Initial state of the system

| | *A* | *B* | *C* | *D* | *E* | *F* | *G* |
|---|---|---|---|---|---|---|---|
| Father | nil | A | A | C | C | E | E |
| Request | F | F | F | F | F | F | F |
| Queue | nil | nil | nil | nil | nil | nil | nil |

*F: False, T: True, nil: empty*

**Table 3.** E requests the critical section

| | *A* | *B* | *C* | *D* | *E* | *F* | *G* |
|---|---|---|---|---|---|---|---|
| Father | nil | A | A | C | C | E | E |
| Request | F | F | F | F | T | F | F |
| Queue | nil | nil | nil | nil | nil | nil | nil |

***C receives the request from E and A receives the request from C:***

As C is not the root, it transmits the request to its father. The table of variables is not changed. A puts E at the end of its local queue. The queue was empty, now it only contains the requesting-site. A is the root, is not in critical section (request = false), then its father becomes C which is the first node on the path to E (Table 4). The quality of rooted tree is lost for a short time. A is the father of C, C is the father of A). A sends the token and the queue to C to inform C that now it will be the new root.

**Table 4.** A has received the request of E

| | *A* | *B* | *C* | *D* | *E* | *F* | *G* |
|---|---|---|---|---|---|---|---|
| Father | C | A | A | C | C | E | E |
| Request | F | F | F | F | T | F | F |
| Queue | E | nil | nil | nil | nil | nil | nil |

**Table 5.** C has received the token from A

| | *A* | *B* | *C* | *D* | *E* | *F* | *G* |
|---|---|---|---|---|---|---|---|
| Father | C | A | nil | C | C | E | E |
| Request | F | F | F | F | T | F | F |
| Queue | nil | nil | E | nil | nil | nil | nil |

***C receives the token and becomes the root:***
C puts E in its local queue. The arrow “C to A” is now inverted. C is a temporary root (Table 5 and Figure 3). After this, the token is transmitted to E and the father of C becomes E. During a short time, the quality of rooted tree is lost (E is the father of C, C is the father of E). C sends the token and the queue to E.

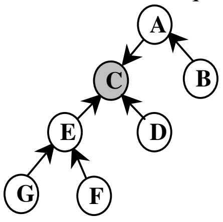


**Fig. 3.** C gets the token

***E receives the token, becomes the root and enters its critical section:***
The arrow “E to C” is now inverted. E becomes the root (Table 6 and Figure 4). C enters its critical section. If another site requests the critical section, the request will be transmitted to E.

**Table 6.** E receives the token

| | *A* | *B* | *C* | *D* | *E* | *F* | *G* |
|---|---|---|---|---|---|---|---|
| Father | C | A | E | C | nil | E | E |
| Request | F | F | F | F | T | F | F |
| Queue | nil | nil | nil | nil | E | nil | nil |

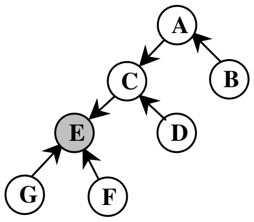


**Fig. 4.** E receives the token

## 4 Algorithm’s Specification

The algorithm is presented as a series of procedures: "Requesting critical section", "Receiving request", "Receiving token", "Release critical section". A site can enter the critical section in "Requesting critical section" and "Receiving token". The procedure "Check the queue" details what to do after releasing the critical section. Precisely a site must check if another site has requested, and, in this case, it must send him the token.

```
Variables of site i

Const me =…{ identity of the site }
      N =… { total number of sites }
Type  Site = 1,…, N ∪ {nil} { nil means indefinite }
Var   Request: Boolean  {says if the site has requested}
      Father: site{Gives the organization of the rooted tree}
      Q-head, requesting-site: Site  {temporary variables}
      Routing: Array [Site] {Routing [j] is the first node on
                               the path from i to j}
      Queue: ordered set of [Site];
Type of messages   Req {transmission of a request}
                   Token {Transmission of the token}
Procedure Initialization
Begin
      Father := …{ nil for the root, else father }
      Request := false;
      Routing :… { the routing table is different for every
                   node}
      Queue :=  nil;
End
General procedures
      Put off (queue) { Put a site off the queue}
      Chain (queue, requesting site)
      { It is the  concatenation of a queue and a site}
Procedure Requesting critical-section
Begin
      Request := True
      If  (father = nil)  then
      begin
            Queue := {me}
            PERFORM CRITICAL SECTION
            RELEASE CRITICAL SECTION
      end
      Else
            Send (req, me) to father
      Endif
Endprocedure
Procedure Receiving request (req, requesting-site)
Begin
      If father = nil) then
            Begin
                  queue := Chain (queue, requesting-site)
                  If (not request) then {the root is not in
                        its critical section}
                  begin
                        father:= Routing (requesting-site)
                        Send (token, queue) to Routing (re-
                        questing-site)
                        Queue := nil
                  endif
            end
      else {The site has only to transmit the request}
            Send (req, requesting-site) to father
      endif
Endprocedure
```

```
Procedure Release critical-section
Begin
      Request := False
      Put off (queue)  {I have finished with the critical
                         section}
      CHECK THE QUEUE (queue)
Endprocedure

Procedure Check the queue (queue)
Begin
      If (not (queue = nil)) then
      Begin
            Q-head := head (queue)
            Father := Routing (Q-head )
            Send (token, queue) to father
            Queue := nil
Endprocedure

Procedure Receiving token (token, received-queue)
Begin
      Queue := received-queue
      father := nil
      If  me = head (queue) then
      Begin
            PERFORM CRITICAL SECTION
            RELEASE CRITICAL SECTION
      End
      else
            CHECK THE QUEUE (queue)
      endif
Endprocedure
```

## 5 Mutual Exclusion and Liveness Are Satisfied

Only a site owns the token and is authorized to send it to another one. That ensures mutual exclusion. The queue is FIFO. When a request is arrived in the queue, we are ensured that the corresponding site will obtain the critical section. It remains to check that every request arrives to the queue. That is not difficult because we are in a tree: there is no cycle. There is only a problem: a request can go from A to B during the time when token goes from B to A. That means the request does not take the shortest path, nevertheless it will arrive in the queue because the communications are two times faster than the passage in critical section (see hypotheses).

## 6 Study of Performances

Let us count the average number of messages in the case of one group for a particular rooted tree. The height of a node i is defined as the number of nodes from i to the root (i and the root included). This means the height of the root is 1. If the root requests the critical section, no messages are exchanged. When another node i, of height $h_i$, requests the critical section, the number of messages is 2 $(h_i -1)$, i. e. $(h_i -1)$ to transmit the request to the root, and $(h_i -1)$ messages to transmit the token from the root to i. Suppose the probability of requesting the critical section, for node i, is $p_i$.
**Lemma 1 :** the average number of messages in a given rooted tree is:$2 \quad p_i (h_i \quad 1)$

**Note** If j is the root, $(h_i -1)$ is equal to d (i, j) where d is the distance between the 2 nodes.
**Corollary:** the average number of messages in a given rooted tree is $2 \quad p_i d(j,i)$ .

e.g., the average number of messages for Figure 5c is $2p_1+4p_3+4p_4+4p_5$.
It has been seen that, when the token has arrived at the requesting site, the tree is not changed, but there is another root. For instance, if 1 requests the critical section, the new rooted tree will be that one of Figure 5b. For a given tree, whatever the root, if i requests the critical section, the graph becomes a rooted tree of root i.

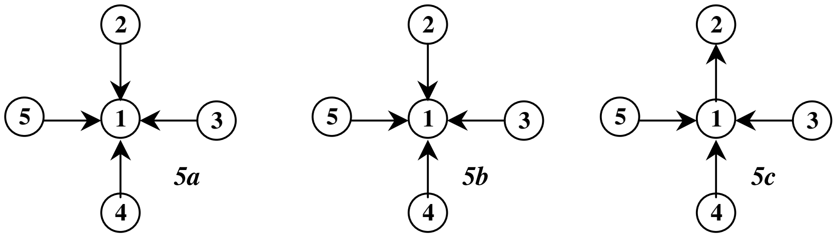


**Fig. 5.** Requests of 1 and 2.

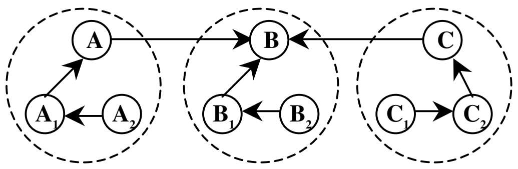
This gives:
**Lemma 2** For a given tree, the probability that the rooted tree of root i appears, is $p_i$. Then, this implies that the average number of messages of a tree will be the sum of the products of the probabilities of the rooted trees by the average numbers of messages:

**Lemma 3** The cost (average number of messages) of a tree is: $2 \overset{n}{\underset{i=1}{}} \overset{n}{\underset{j=1}{}} p_i p_j d(i,j)$ **.**

We have proved [3] that the cheapest tree is the star with the greatest probability at the center. The cost is less than 4 messages. However, star is not a good structure for fault tolerance, because the center is frequently impelled. Raymond obtains an average number of messages of O(Log (n)) by simulation, in the case of equiprobability.

## 7 Algorithm for n Groups

Maekawa [6] has written an algorithm by groups without priority. For us, groups are disjoint and every group has a priority level. When there are some requests in the queue, they are sorted in function of priorities. There is a router by group. The same structure (rooted tree) is defined between the routers than between the sites of a group. And the algorithm between the groups is nearly the same than between the routers of a group. The token is owned by the site at the root of the group, whose the router is at the root. When a site requests the critical section, it sends its request towards the root of its group. If the token is in the group, the request is put in the queue. A reorganization of the queue is processed in function of the site's request's priorities. When the site in critical section releases it, the token is sent to the first site of the queue (this site has the maximum priority). If the token is not in the group, the request is sent to the router of the group, which sends it to the group owner of the token (figure 6). The detailed program is given in [3].



*- A, B, C are routers,*
*- $A_1$, $A_2$ are sites of group A,*
*- $B_1$, $B_1$ are sites of group B,*
*- $C_1$, $C_2$ are sites of group C.*

**Fig. 6.** Example for three structured groups.

## 8 Mutual Exclusion Is Satisfied

The proof of mutual exclusion is the same than in one group. Concerning deadlock, it is the same thing: deadlock is impossible. Nevertheless, there is absence of equity in prioritized mutual exclusion, because it is the aim of the system: it is possible that reordering the queue prevents a site to obtain the critical section.

## 9 Performances

There are thee kinds of messages : messages inside a group, messages from a router to another one, and messages between the routers and the sites. If the structure of group is a star, the cost is less than 4 messages. It will be the same for the communications between the routers. The result is that with a star for each group and a star between the routers, the number of messages is less than 12. That is really less than 12 if there are particular groups with great probabilities. Heterogeneity is a good factor of economy. Nevertheless, as for one group, the star is not a good structure for fault tolerance, because the center is frequently impelled.

## 10 Conclusion and Perspectives

The advantage of this algorithm is its simplicity, for one or many groups. A first objective is to validate the program. We think a proof with a validation tool will be simpler to obtain than with a mathematical proof. Concerning the performance, it is the same thing: performance by simulation will be simpler to obtain than mathematical performance. We have only considered the number of messages as measure of performance. We will have to consider also the efficiency for fault tolerance. That will give us completely different approach. There is always a problem concerning prioritized problems. It is possible that a site which requests critical section cannot obtain it. It would be interesting to give him the critical section after a maximum number of other sites. We intend to rewrite a new algorithm, to satisfy this specification.

## References

1 A. Goscinski. "Two algorithms for mutual exclusion in real-time distributed computer systems", The Journal of Parallel and Distributed Computing, 9: pp.77-82, (1990).

2 K. Harathi and T. Johnson, "A priority synchronization algorithm for multiprocessors", Technical Report tr93.005. Available at ftp.cis.ufl.edu:cis/tech-reports, (1993).

3 A. Housni, M. Tréhel, "Specification of the prioritized algorithm for N groups", intern paper, Laboratore d'Informatique de l'université de Franche Comté, France, February (2001).

4 T. Johnson, R. E. Newman-Wolfe "A comparison of fast and low overhead distributed priority locks", Journal of Parallel and Distributed Computing 32 (1): pp.74-89 (1996).

5 B. M. K. Qazzaz "A new prioritized mutual exclusion algorithm for distributed systems", Doctoral Thesis, Dept of Comp. SCI., Southern Illinois University, Carbondale, (1994).

6 M. Maekawa, "A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems," ACM Transactions on Computer Systems, Vol3, pp. 145-159 (1985).

7 F. Mueller "Prioritized token-based mutual exclusion for distributed systems", 12th IPPS/SPDP, Orlando, Florida USA, (1998).

8 K. Raymond, "A tree-based algorithm for distributed mutual exclusion" ACM Transactions on Computer Systems Volume 7, Issue 1, pp.61-77 (1989).

9 M.Tréhel, M. Naimi, "A distributed algorithm for mutual exclusion based on data structures and fault-tolerance", 6th annual IEEE conference on computers and communications, Phoenix, Arizona, February (1987).