Group Mutual Exclusion - Role Processes

Karina M. Cenci and Jorge R. Ardenghi

Laboratorio de Investigación en Sistemas Distribuidos
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
{kmc,jra}@cs.uns.edu.ar

Abstract. This paper presents an extension to group mutual exclusion (GME) where processes join a group with a role (shared, exclusive) in each stage. The properties that must guarantee a solution to GME are: mutual exclusion, bounded delay, progress and concurrency. For this new situation, a new property role mutual exclusion is introduced. A solution is proposed in a network with no share memory whose members communicate by messages. The proposed algorithm is composed of two players: groups and processes, where groups are passive players and processes are active players. For the coordination access to the resource, each group has been assigned a quorum.

Keywords: Mutual Exclusion - Group Mutual Exclusion - Concurrency - Distributed Systems

1 Introduction

Applications use resources to give users service. These applications may require the exclusive use of resources. The use of protocols that guarantee mutual exclusion provides a solution of this problem. Several authors proposed different options using the shared-memory or message-passing model ([1], [2], [10], [3], [8]). In distributed systems, applications need resources too, but there may be two different options: some processes compete and some processes collaborate to give users service. There may also be another situation where a resource can be shared by processes with common property, i.e., they belong to the same *group* or they will not be in conflict while using the resource. Processes with different properties must use the resource in exclusive mode. For example, database applications require a mutual exclusion property for data inserting operation and concurrency property for data selecting operation. This situation is similar to the readers-writers problem (multiple readers-single writer). This type of problem is solved by using protocols that guarantee group mutual exclusion (GME). Properties of mutual exclusion and concurrency are important at the time of the design.

The GME problem was first presented by Joung [6]. The solution presented was an asynchronous algorithm for shared memory parallel computer system. Several quorum-based algorithms [7] [12] [9] have been proposed for asynchronous message passing. The Manabe-Park [9] algorithm prevents the unnecessary blocking, defined as the case in which two processes are prevented from entering a critical section simultaneously even

if they are capable of doing so. Singh-Su [11] proposed a solution to the region synchronization problem (such as mutual exclusion, group mutual exclusion, readers/writers) using messages and satisfying the property of absence of unnecessary blocking. In [5], the proposed algorithm is a distributed solution to the problem of group mutual exclusion coordination, considering that the processes require some time to share the resource in a group with a maximum concurrency of n processes.

In this paper, we present an extension to the problem of group mutual exclusion. The process selects a role (*shared*, *exclusive*) to join the group in a stage. For this situation, a new property is introduced: role mutual exclusion.

2 Preliminaries - Base Model

Let be a set of n processes P_0, P_1, ..., P_{n-1}; a set of m groups G_0, G_1, ..., G_{m-1} and a unique, non shareable, resource among the m groups. The processes may work alone or in cooperation with other processes in a group. Any of the n processes is able to participate in a group. Only one group at a time is allowed to use the resource.

Initially each process works alone. When the process wants to work in a team, it selects a group. Each process may select any of the different groups with a finite working time in the team. Figure 1 shows an example of the relation between the groups and the processes. Where P_1, P_2 and P_7 are linked to the group G_1, the latter is active and has the permission to use the resource. That means that all the processes are using the resource concurrently. Processes P_0 and P_8 are linked to the group G_0 that is competing to gain access to the resource.

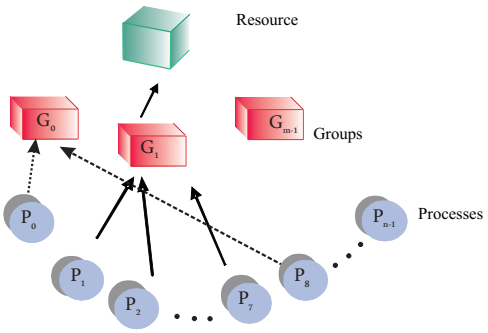


Fig. 1. Example of Relation between the players

The model of two players, posed in [4], proposed a general solution to this problem using two players: *groups* and *processes*. Figure 2 shows the communication between the players. The *processes* are active players and the *groups* are passive players. The relation between the players is temporary. When the player group is activated, the competition to access the resource begins.

The design of a solution for this problem requires an algorithm that satisfies the followings requirements.

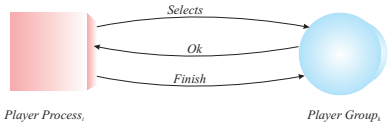


Fig. 2. Communication between the players

- Mutual Exclusion: if some process is in a group, then no other process can be in a different group using the resource simultaneously.
- Bounded Delay: a process attempting to participate in a group will eventually succeed.
- Progress: when the resource is available (the critical section is empty), and some groups are waiting, one group gains access to the resource at some later point.
- Concurrent Entering: if some processes are interested in a group and there is no other process interested in a different group, the processes can participate in the group concurrently.

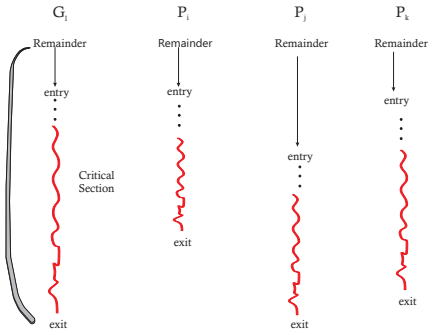


Fig. 3. Concurrency in group g

Figure 3 shows an example of concurrency among processes. Group G_l is in the critical section; P_i, the first associated process to the group, and P_k access together to the critical section. At the moment process P_j selects group G_l, the latter is in the critical section and the first associated process (P_i) is still working. Then process P_j can access without waiting and work concurrently.

3 Definition of GME-RP

This section presents a variation of GME defined as GME-RP (Group Mutual Exclusion - Role Process).

Let be a set of n processes P_0, P_1, ..., P_{n-1}; a set of m groups G_0, G_1, ..., G_{m-1} and a unique, non shareable, resource among the m groups. The processes may work

alone or in cooperation with other processes in a group. Every time that a process works in cooperation, it selects a role to participate in the group.

The roles are *exclusive* and *shared*. The *exclusive* role means that a process requires some time to use the resource exclusively with respect to other *exclusive* processes of the same group and concurrently with *shared* processes. Examples that correspond to this situation are the following. (1) In a virtual class, students participate in shared role in the class and tutors in exclusive role. The virtual class is the group that all members want to join. Some processes access the virtual class (resource) as student role (shared role) and others as tutor role (exclusive role). (2) In a political debate, candidates participate in exclusive role and the public in shared role. The political debate is the group that all members want to join.

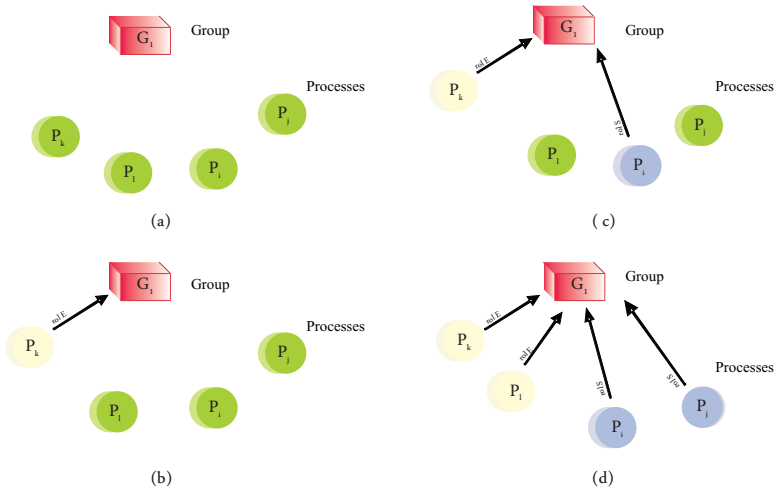


Fig. 4. Model

Initially, each process works alone. Figure 4 (a) shows this situation. When the process wants to work in a team, it selects a role and a group. Each process may select any of the different groups with a finite working time in the team. The selected role depends of the activities of the process in the selected group for this stage. Figure 4 (b) shows the case in which process P_k is linked to group G_1 with *rolE* (*exclusive* role).

Figure 4 (c) shows process P_k with *rolE* and process P_i with *rolS* (*shared* role) associated to group G_1. Figure 4 (d) shows four associated processes to group G_1, where two processes, P_k and P_l, with *rolE* and two processes, P_i and P_j, with *rolS*.

A solution to this problem requires an algorithm that satisfies the requirements of GME:

- Mutual Exclusion
- Bounded Delay
- Progress
- Concurrent Entering

– Role Mutual Exclusion

and the following,

– Role Mutual Exclusion: means that some process with exclusive role is in a group, then no other process with exclusive role in the same group can be using the resource simultaneously.

4 Model GME-RP

This section presents a model to solve the problem of GME-RP. The proposed pattern is based on the model of two players. The behaviour of the players is introduced first and then, a design of an algorithm applying these concepts.

The behaviour of the player process is the following:
- When the player process wants to participate in a group, it first specifies its role, time and finally the group.
- The process waits until the group allows the access.
- When the process finishes, it realeses the group.

The behaviour of the player group is the following:
- The moment the player group becomes active, a time to use the resource is assigned to it. This time is that of the first process.
- While the player group is waiting to access the resource (entry section)
  . A request from a player process may arrive. The request is added to the active queue. The group checks the process role of the request.
  If it is *shared*, the group compares the duration of the process with its own duration. If it is longer, it then sets the associated time to the maximum duration of the new player process.
  If it is *exclusive*, the group adds the duration of the process to its associated time.
- While the player group is using the resource (critical section):
  .A request from a player process may arrive. The group checks the process role of the request.
  If it is *shared*, the duration of this process is compared to the remainder (group duration - elapsed duration). If it is not longer the group then adds the request to the active queue and accepts it. Otherwise, this request is added to the waiting queue until the next stage.
  If it is *exclusive*, the duration of this process is added to the sum of the duration of the requests with exclusive role of the active queue, and the total is compared with the remainder (group duration - elapsed duration). If it is not longer the group then adds the request to the active queue and accepts it. Otherwise, this request is added to the waiting queue until the next stage.

When the player group is in a critical section and a new request arrives, it does an acceptance test (1).

$$\begin{cases} (a)\ Shared\ Role & tpodur_{nreq} \leq timegroup - tpouse \\ (b)\ Exclusive\ Role & \sum_{Req_{j,E} \in LP} tpodur_j + tpodur_{nreq} \leq timegroup - tpouse \end{cases} \quad (1)$$

where $Req_{j,E} \in LP$ means all the accepted requests with exclusive role to use the resource in this stage.

5 Algorithm GME-RP

The algorithm presented uses messages for the communication. In a distributed environment, we have to consider the communication time (delay time). We assume a reliable network, with an estimated communication time *tc*, and a finite resource time use. The communication time is necessary to adjust the remainder time, to accept or not a new player process while the player group is in the critical section. We define the following variables:

- $tc_{i,k}$: Delay estimation of the communication between group G_k and process P_i
- $tpodur_i$: Process time associated to the group in a stage
- $role_i$: Process role associated to the group in a stage
- $gtpo_k$: Group time in a stage

Considering the defined variables, the acceptance test is the following:

$$\begin{cases} (a)\ Shared\ Role & tpodur_i \leq remaindertime_k - tc_{i,k} \\ (b)\ Exclusive\ Role & \sum_{Req_{j,E} \in LP} tpodur_j + tpodur_i \leq remaindertime_k - tc_{i,k} \end{cases} \quad (2)$$

where remaindertime$_k$ = (gtpo$_k$ - tpouse$_k$).

The process, in each stage, sends two messages to the group and receives one message from the group.

- Req-Process (G_k, P_i, tpodur$_i$, role$_i$): process P_i sends a request message to group G_k to participate during a period tpodur$_i$ with role$_i$.
- Rep-Process (G_k, P_i): process P_i receives the reply to its request from G_k, that allows access to the resource.
- Rel-Process (G_k, P_i): process P_i sends a message to group G_k to inform that the period of time in the group has finished and it is unlinked.

The variables of group G_k are the following: *state* (INACTIVE, ACTIVE, CS, EXIT), LP: keeps information of all linked processes, LW: keeps information of all waiting processes for the next stage, LG: keeps information of all waiting requests of *lock*, gtpo$_k$: keeps the time to use the resource. The group communicates with the processes and with other groups. The messages received from the process are:

- Req-Process(G_k, P_i, tpodur$_i$, role$_i$) this message is received from a process. If the group is INACTIVE then it changes its state to ACTIVE, adds the request to the LP list and adjusts the group time (gtpo$_k$) to the process time (tpodur$_i$).
  If the group is ACTIVE the request is added, and obtains the request role. If the role is *exclusive*, the group adds the duration of the process to its associated time. If the role is *shared*, the group time is compared with the process time. If it is shorter then it adjusts its current time to the process time.
  If the group is in CS it obtains the request role. If the role is *shared*, the group checks its remainder time with the process time. If it is longer, the group accepts

the request process, allows it to participate in this stage, and adds the request to the LP list. Otherwise, the process has to wait for the next stage and adds the request to the LW list. If the role is *exclusive*, the group checks its remainder time with the sum of all pending requests duration with exclusive role in LP plus the new request process duration. If it is longer, it accepts the process request, permits it to participate in this stage, and adds the request to the LP list. Otherwise, the process has to wait for the next stage and adds the request to the LW list.

- Rel-Process(G_k, P_i) this message comes from a process to release its link with the group. The latter, obtains the request role and removes the request from the LP list. If the LP list is empty, the group releases the resource. If there are waiting processes in the LW list then the group begins a new stage.
  If the LP list is not empty and there are pending requests with exclusive role, the group allows one of them to access.

The messages received from the other groups are:

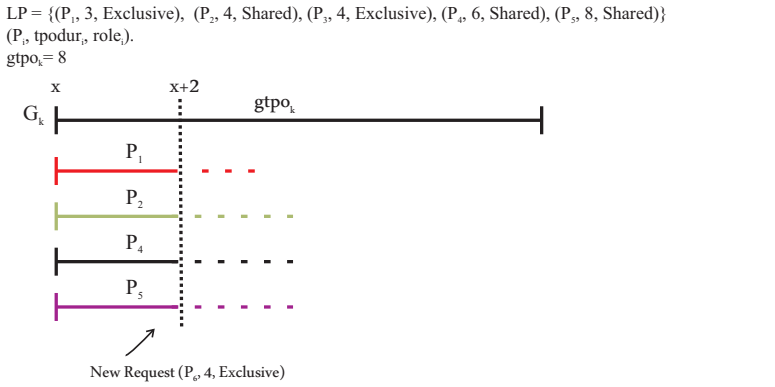
- Req-Group(G_l, priori) this message comes from group G_l that requires the *lock*. The group G_k grants the lock if available. If the lock is not available two different cases may occur: (a) The priority of the received message is less than the priority of the message the lock has been given to. In this case the request is delayed. (b) If the priority is higher it requests the lock from the appropriate group and grants it the highest priority.
- Rec-Group(G_l, G_k) this message comes from group G_l, affirmative response to the message *Req-Group* of requirement lock. If group G_k has all the locks,it changes the state to CS, informs all the processes with *shared* role that are linked to the group and allows the first request with *exclusive* role to access.
- Rel-Group(G_l, G_k) this message comes from group G_l requiring the lock. This will be successful if group G_k is not in the critical section.
- Rep-Rel-Group(G_l, G_k) this message comes from group G_l releasing the lock that was given to group G_k. The lock is granted the highest priority requirement.
- Lib-Group(G_l) this message comes from group G_l that releases the lock. If there are outstanding requirements, it chooses the highest priority and grants the lock.

Figure 5 shows group G_k in CS, with its linked processes. The LP list shows the processes ordered according to their arrival time. Processes P_1, P_2, P_4 and P_5 are concurrently in CS. At $x + 2$, a new request from process P_6 with exclusive role and $tpodur$=4 arrives. Since the sum of the time of all exclusive requests is greater than $(gtpo_k - tpouse_k - t_{7,k})$ and the group is in the CS the process P_7 has to wait for the next stage.

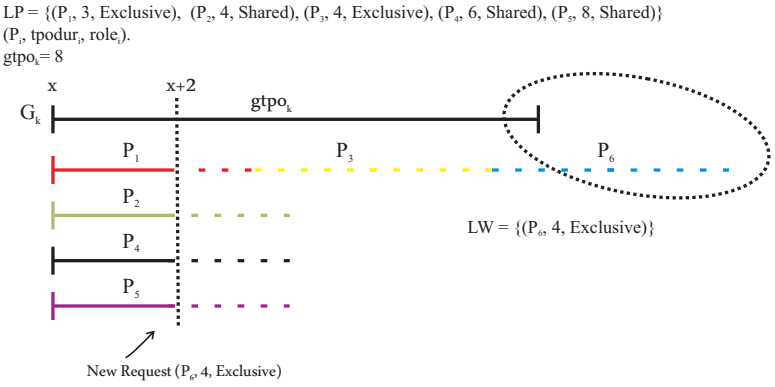
The messages among the groups correspond to the competition to gain access to the resource. The algorithm uses messages to obtain permission from the other groups. Each group has an associated quorum (set of groups) to request permission of access. To select the quorum, we use the Maekawa method [8]. The size of the quorum is $\sqrt{m}$, where m is the number of groups. When the group obtains all the permissions the resource can be used and this is informed to its associated processes.

5.1 Correctness and Complexity

In this section, we show the correctness of the proposed algorithm. The algorithm satisfies the properties of mutual exclusion, progress, concurrent entering and role mutual exclusion.



(a) Group *k* CS - Request from P_6



(b) Group *k* CS - Request from P_6 has to wait

Fig. 5. Group and Processes

Theorem 51 *Role Mutual Exclusion satisfied the proposed algorithm.*

For role mutual exclusion to be achieved, at most one process with *exclusive* role should enter the critical section at any time. If group G_k is in CS and allows process P_i with exclusive role to access, then it waits until P_i finishes to allow another process (P_j) with exclusive role to access.

Theorem 52 *The maximum number of processes associated to a group is* n.

When each process makes a request for the same group simultaneously, all the requests are added to the active queue. When the group grants the locks of its quorum, it can access all the processes with *shared* role and one process with *exclusive* role concurrently. If there is only one request with exclusive role then it can access the n processes concurrently.

Theorem 53 *The proposed algorithm ensures bounded delay.*

Suppose a process P_i makes a request to group G_k and is waiting indefinitely. This means that:

(a) Another group (G_l) stays indefinitely in the critical section. This situation would happen if the arrival of all new processes are accepted. This is not possible because, when a new request arrives and G_l is in critical section, the group performs the acceptance test. If the test is fine, the group accepts the request for this stage. Otherwise, it waits for the following stage.

(b) G_k waits indefinitely to access the critical section. Each group request has an associated priority (G_k, priori). This priority will eventually be the highest and grants the access to the critical section.

The complexity of an algorithm can be measured using different topics, like the number of access to shared memory, the delay time between entries in the critical section and the number of exchanged messages. The election of the measure depends on the type of the algorithm. For this algorithm the complexity is measured in function of the number of the messages requires. Let $q = |S_k|$. In the best case: If the group has one associated process, it will require $3+3(q-1)$ messages; if it has l associated processes, it will require $3l+3(q-1)$ messages. With the maximum number of processes associated, n requests for the same group simultaneously, the algorithm requires $3n + 3(q - 1)$ messages. If in average, each group has to yield once, the number of messages required is $5(q-1)$ to grant the permission. If each group has to yield the permission at most p times, then it requires $3l + 3(q - 1) + 2p(q - 1)$ messages with l associated processes.

Conclusions

In this paper we proposed an extension of the group mutual exclusion problem, called group mutual exclusion role process (GME-RP). The model for this problem considers that processes have a role and an associated time to share the group. This should be the time they will work cooperatively in the group in each stage. Designing a solution to this problem requires satisfying the properties of GME and the role mutual exclusion property.

A solution is presented using messages for the communication among processes and groups. The groups have been assigned a quorum, that is used in the competition to get the permission to access the resource. The algorithm guarantees the properties and in the best case, with l processes linked, it requires $3l + 3(q - 1)$ messages.

References

1. J. Anderson and Y. J. Kim. Adaptive mutual exclusion with local spinning. *Proceedings of the 14th International Symposium on Distributed Computing*, October 2000.
2. H. Attiya and V. Bortnikov. Adaptive and efficient mutual exclusion. *Distributed Computing*, 15(3):177–189, 2002. Proceedings of 19th Annual ACM Symposium on Principles of Distributed Computing, Julio 2000.
3. D. Barbara and H. García-Molina. Mutual exclusion in partitioned distributed systems. *Distributed Computing*, 1:119–132, 1986.
4. K. M. Cenci and J. Ardenghi. Modelos dos actores para grupos de procesos. In *XIV Congreso Argentino de Ciencias de la Computación (CACIC 2008)*, 2008.
5. K. M. Cenci and J. R. Ardenghi. Group mutual exclusion based on priorities. *JCS&T*, 11(1):21–26, April 2011.
6. Y. J. Joung. Asynchronous group mutual exclusion (extended abstract). In *Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing (PODC'98)*, pages 51–60, June 1998.
7. Y. J. Joung. Quorum-based algorithms for group mutual exclusion. *IEEE Transactions on Parallel and Distributed Systems*, pages 463–476, May 2003.
8. M. Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, 3(2):145–159, May 1985.
9. Y. Manabe and J. Park. A quorum-based extended group mutual exclusion algorithm without unnecessary blocking. In *Proceedings of th Tenth International Conference on Parallel and Distributed Systems*, pages 341–348, 2004.
10. M. Raynal. Algorithms for mutual exclusion. Technical report, MIT Press, Cambridge, MA, 1986.
11. Gurdip Singh and Ye Su. Efficient synchronization in message passing systems. In *22nd International Conference on Advanced Information Networking and Applicaciones*, pages 219–226, 2008.
12. M. Toyomura, S. Kamei, and H. Kakugawa. A quorum-based distributed algorithm for group mutual exclusion. In *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 742–746, 2003.