View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Servicio de Difusión de la Creación Intelectual

Access Coordination: Group of 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. We propose a distributed algorithm for the group mutual exclusion problem in a network with no share memory whose members only communicate by messages. The proposed algorithm is composed by two players: groups and processes, groups are passive players while processes are active players. For the coordination access to the resource, each group has assigned a quorum. The groups have associated priorities in each stage, meanwhile the processes have the same level priority. An important feature is that processes have associated a time to participate in the group in each stage.

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

1. Introduction

In distributed systems there are processes that compete in using resources and others that cooperate and share resources for solving a task. The main problem to solve is to recognize it as a mutual exclusion one. This problem arise in multiprogramming environments because processes require exclusive access for using resources, e.g. printers, database. Differents solution have proposed to solve this problem, e.g. [2], [3], [10], [11], [12]. When some processes cooperate and others compete appears a difference of the original problem, knowed as group mutual exclusion, there are differents approach for this problem using differents paradigms and implementations, e.g. [4], [5], [7], [8], [13]. In the new approach two properties are important: exclusion among competing processes and concurrence among cooperative processes.

In this paper, we propose a distributed algorithm to the problem of group mutual exclusion coordination, considering that the processes require a time to share the resource in a group. We consider that every group has an associated priority.

2. Preliminaries

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 others 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, selects a group. Each process may select any of the differents groups with a finite time of work in the team. Figure 1 shows an example of relation between the groups and the processes; where p_1, p_2 and p_7 are linked to the group g_3, this is active and has the permission to use the resource, that means that all the processes are using concurrently the resource. Processes p_0 and p_8 are linked to the group g_3 that is competing to gain the access to the resource.

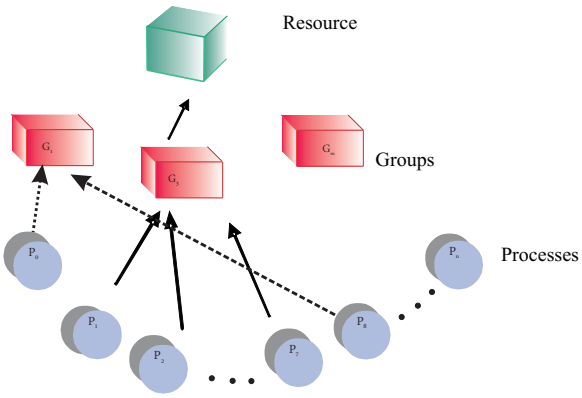

Fig. 1. Example of Relation between the players

The model of two players, posed in [6], 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, begins the competition to access the resource.

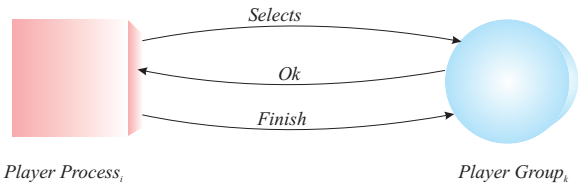

Fig. 2. Communication between the players

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

- Mutual Exclusion: if some process is in a group, then no other process can be in a different group 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, at some later point one group gain the access to the resource.
- Concurrent Entering: if some processes are interested in a group and no process is interested in a different group, then the processes can participate in the group concurrently.

3. Algorithm GPTP (Group Priority Time Process)

This section presents a solution to the problem of group mutual exclusion including time associated with the players (groups and processes), using messages for the communication. Applying the model of the two players, [6], to this situation, we obtain the following:

- When the player process wants to participate in a group, first specifies his time and then selects the group. Waits until the group allow the access.
- At the moment the player group activates, its assigns the time (duration) of the first process to use the resource.
  While the player group is waiting to access to the resource (entry section):
  When a request from a player process arrives, adds the request to the active queue and compares the duration of the process with de group duration, if it is greater, then sets the duration to the maximal duration of the new player process.
  While the player group is using the resource (critical section):
  When a request from a player process arrives, compares if the duration of the process in not greater than remainder duration (group duration - elapsed duration) then adds the request to the active queue and accept the request, otherwise adds the request to the waiting queue until the next stage.

The time associated with each player does not represent deadline time but represents its duration in the critical section. In distributed environment, we have to consider the communication time (time delay). We assume a reliable network, with a estimated communication time tc, and a finite time of use of the resource. The communication time is necessary to adjust the remainder time (duration), for 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 the group G_k and the process P_i

– $tpodur_i$: Process time associated to the group in a stage
– $gtpo_k$: Group time in a stage

When the player group receives a request from a process and it is in the critical section, the acceptance control for a new process is the following: $\text{tpodur}_i < (\text{remainder time}_k - \text{tc}_{i,k})$, where $\text{remainder time}_k = (\text{gtpo}_k - \text{tpouse}_k)$

When the time group finishes, we consider the following options:

1. Waits until all the associated processes release the group.
2. Inform the associated processes in order to finish theirs associations. We could consider different possibilities.
   (a) Waiting for all of the process acknowledge. The delay time could be unpredictable.
   (b) Release the critical section (release the resource) and allow another group to access. This option avoids the waiting time, but the notification could be delay to a associated process and continue using the resource while a different group gain the access to the critical section. This situation no guarrantees the group mutual exclusion property.

In accordance with the constrains imposed for allowing the active association in the group, we assume that the group does not take in acount its own time, when each process finish that, informs to the group; when the group is empty of participants processes release the critical section, this option simplifies the solution and is acceptable because the associated time is not critical.

The figure 1 shows the actions of the player process. The process, in each stage, sends two messages to the group and receives one message from the group.

process P_i
  RemainderSection
    ...
  EntrySection
    G_k = chosen group
    tpodur_i = chosen the time to use the resource
    send Req-Process (G_k, P_i, topdur_i)
    receive Rep-Process (G_k, P_i)
  CriticalSection
    ... *duration* $tpodur_i$
  ExitSection
    send Rel-Process (G_k, P_i)

Fig. 3. Actions of Player Process P_i

– Req-Process (G_k, P_i, topdur_i): the process P_i sends a request message to the group G_k to participate in during a period topdur_i.
– Rep-Process (G_k, P_i): the process P_i receives the reply of his request from G_k, that allow to access the resource.

– Rel-Process (G_k, P_i): the process P_i sends a message to the group G_k to inform, it has finihed the period in the group and it is unlink.

The time that the process stays in the critical section is $tpodur_i$ and then release its association with the group.

The figure 4 shows the variables of the group G_k. The figure 5 shows the actions of the player group associated with the process and the others groups. The states of the group are the followings: INACTIVE: is waiting for participating processes, ACTIVE: is waiting to access the resource, CS is using the resource and EXIT is releasing the resource. Each linked process has the same priority, and each group has an associated priority, two different groups could have the same priority. The proposed protocol is based on priority without prompt meanwhile the group with lower priority is using the resource.

Variables

state (INACTIVE, ACTIVE, CS, EXIT)
LP: keeps information of all the linked process.
LG: keeps information of all the pendent request of *lock*.
$gtpo_k$: keeps the time to use the resource.
priori: keeps the priority of the group in the stage.

Fig. 4. Variables Group G_k

The group communicates with the associated processes and with the others groups. Messages received from the process are:

– Req-Process(G_k, P_i, $tpodur_i$) this message is received from a process, if the group is INACTIVE then changes its state to ACTIVE, adds the request to the list LP and sets the length of time of the group ($gtpo_k$) with the length of time of the process ($tpodur_i$). If the group is ACTIVE adds the request, and checks the length of time of the group with the length of time of the process, if it is the lowest then sets its current time length with the process time length. If the group is in SC adds the request and checks the remaining time group with the length of time process, if it is the greatest then accepts the process request and allow to participate in this stage, otherwise the process has to wait for the next stage.
– Rel-Process(G_k, P_i) this message comes from a process to release his link with the group. Removes the request from the list LP. If the list LP is empty of active process then release the resource. If exit waiting processes in the list then the group begins a new stage.

Figure 6 (a) shows the state of the group g_k (ACTIVE) with their linked processes p_i, p_j and p_m, the time of the group is equal to the time of process p_m. Figure 6 (b) shows when arrives a new request from process p_s to link the group with a time ($tpodur_s > gtpo_k$), since the group is in the ACTIVE state, sets the value of its time with the time of p_s, figure 6 (c) shows this modification.

group G_k

♢ Receive Req-Process(G_k, P_i, $tpodur_i$)

case state of

"Inactive": $gtpo_k = tpodur_i$; state = "Active";

conj = ∅; priori = chosen priority; AddLp(LP, P_i)

AddLG(LG, G_k, priori); send multicast Req-Grupo(G_k, prior)

"Active": if $tpodur_i > gtpo_k$ then $gtpo_k = tpodur_i$

AddLp(LP, P_i)

"SC": AddLp(LP, P_i)

if $tpodur_i \leq (gtpo_k - tpouse_k - tc_{i,k})$ then send Rep-Process(G_k, P_i)

"Exit": AddLp(LP, P_i)

♢ Receive Rel-Process(G_k, P_i)

DeleLp(LP, P_i)

if activeempty(LP) then

state = "Exit"; send multicast Lib-Group(G_k); G_l = SelectGroup(LG)

send Rec-Group(G_l, G_k); state ="Inactive"

if not empty(LG) then

state = "Active"; conj = ∅; priori = chosen priority

send multicast Req-Group(G_k, priori)

♢ Receive Req-Group(G_l, priori)

if empty(LG) then

AddListGroup(LG, G_l, priori); send Rec-Group(G_k, G_l)

else

if HigherPriori(LG, G_l, priori) then G_s = findHigh(LG);

send Rel-Group(G_s, G_k); AddListGroup(LG, G_l, priori)

else

AddListGroup(LG, G_l, priori)

♢ Receive Rec-Group (G_l, G_k)

if $G_l \notin$ conj then

conj = conj ∪ {G_l}

if |conj|=|S_k| then

state = "CS"

For each process in LP do

send Rep-Process(G_k,P_i)

♢ Receive Rel-Group (G_l, G_k)

if state ≠ "CS" then conj = conj - {G_l}; send Rep-Rel-Group(G_k, G_l)

♢ Receive Rep-Rel-Grupo (G_l, G_k)

G_s = findHigher(LG); send Rec-Grupo(G_k, G_s)

♢ Receive Lib-Group (G_l)

DeleListGroup(LG, G_l)

if not emptyListGroup(LG) then

G_s = findHigher(LG); send Rec-Group(G_k, G_s)

Fig. 5. Actions of Player Group G_k

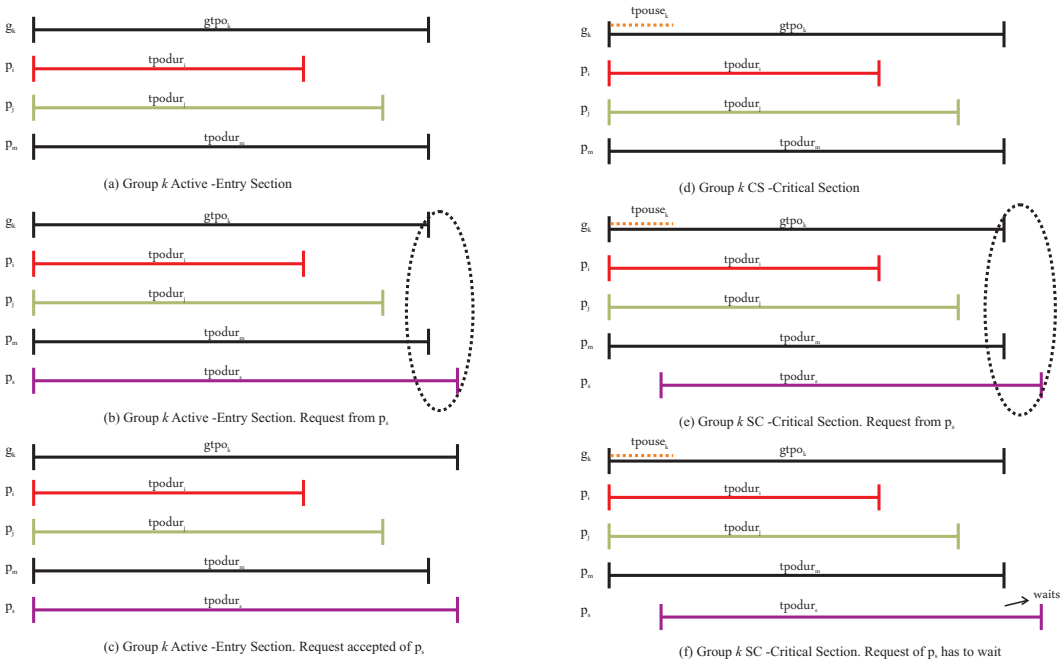

Fig. 6. Concurrency

Figure 6 (d) shows the state of the group g_k (SC) with their linked processes p_i, p_j and p_m, and the $tpouse_k > 0$. Figure 6 (e) shows when arrives a new request from process p_s to link the group with a time $tpodur_s$. Since the time $tpodur_s$ is greater than ($gtpo_k$ - $tpouse_k$ - $tc_{s,k}$) and the group is in SC then the process p_s has to wait for the next stage, figure 6 (e) shows this case.

The messages among the groups correspond to the competition for gain the access of the resource. The algorithm uses messages to obtain the permissions from the others groups. Each group has associated a quorum (set of groups) to request the permission of access (S_k). To select the quorum, we use the Maekawa method [11], the size of the quorum is $\sqrt{M}$, where M is the number of groups. When the group obtains all the permissions can use the resource and inform to his associated processes.

The algorithm satisfies the properties of mutual exclusion, progress and concurrent entering. But has problems and not satisfies bounded delay, this happens because we consider priorities associated to the groups, know as starvation. To minimize this:

- When a group with a lower priority is using the resource, the others groups with higer priority that want to access they must wait.
- When a group is using the resource (state = "SC"), accepts new requests of processes only if the time associated is less than his remainder time. With this consideration, none of the groups stays indefinitely using the resource.

4. Complexity

The complexity of the algorithms can be measure using different topics, like the number of access to shared memory, the delay time between entries in the critical section, the number of exchanged messages. The election of the measure depends on the type of the algorithm.

The complexity of the algorithm is measured in function of the number of the messages requires. In the best case, each group requires for gain the access and release $3(\sqrt{M}-1)$ messages, where $(\sqrt{M}-1)$ for request the permission, $(\sqrt{M}-1)$ for grant the permission, $(\sqrt{M}-1)$ for release the permission. If it has associated: one process, in total requires $3+3(\sqrt{M}-1)$; l processes, in total requires $3l+3(\sqrt{M}-1)$. The worst case could not be estimated, because the algorithm suffers from starvation. If in average, each group has to yield once so, the number of messages requires are $5(\sqrt{M}-1)$, where $(\sqrt{M}-1)$ for yield the permission, $(\sqrt{M}-1)$ for grant the permission, with l associated processes in total requires $3l+5(\sqrt{M}-1)$.

5. Conclusions

In this paper, we proposed a distributed algorithm for group mutual exclusion considering that processes have an associated time to share the group, this should be the duration they will cooperatively work in the group in each stage.

The algorithm is based on priorities over the groups with no prompt, the communication among the processes and groups uses messages. The groups have assigned a quorum, that it uses in the competition for reach the permissions to access the resource.

The algorithm guarantees mutual exclusion, progress and concurrent entering, but suffers of starvation because of the priorities. In the best case, where the group does not yield the permission, with l processes linked, requieres $3l + 3(\sqrt{M} - 1)$ messages. This solution may suffer of starvation, it could be improved through the concept of aging.

References

1. Attiya H., Hendler D., Woelfel P. *Tight RMR Lower Bounds for Mutual Exclusion and other Problems Proc. 27 th. ACM Proceedings on Distributed Computing, pp 447, 2008.*
2. *D. Barbara, H. García-Molina. Mutual exclusion in partitioned distributed systems. Distributed Computing, vol. 1, no. 2, pp. 119–132, 1986.*
3. *Karina Cenci, Jorge Ardenghi Exclusión Mutua para Coordinación de Sistemas Distribuidos. CACIC 2001.*
4. *K. Cenci, J. Ardenghi Algoritmo para Coordinar Exclusión Mutua y Concurrencia de Grupos de Procesos CACIC 2002.*
5. *K. Cenci, J. Ardenghi Exclusión Mutua en Grupos de Procesos a través de Mensajes CACIC 2003.*
6. *K. Cenci, J. Ardenghi Modelos Dos Actores para grupos de procesos CACIC 2008.*
7. *C. Reyes, K. Cenci Modelo Temporizado de Exclusión para Grupos de Procesos CACIC 2006.*
8. *Yuh-Jzer Joung Asynchronous Group Mutual Exclusion (extended abstract). In Proc. 17 th. ACM Proceedings on Distributed Computing, 1998.*
9. *Y. J. Joung. Quorum-Based Algorithms for Group Mutual Exclusion. IEEE Transactions on Parallel and Distributed Systems, pp. 463–476, Mayo 2003.*
10. *L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, Julio 1978.*
11. *M. Maekawa. A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems. ACM Transactions on Computer Systems, vol 3, issue 2, pp. 145–159, Mayo 1985.*
12. *Michael Raynal. Algorithms for Mutual Exclusion. MIT Press, Cambridge, 1986.*
13. *K. P. Wu, Y. J. Joung. Asynchronous Group Mutual Exclusion in Ring Networks. 13th International Parallel Processing Symposium / 10th Symposium on Parallel and Distributed Processing (IPPS / SPDP '99), 12-16 Abril 1999. Proceedings. IEEE Computer Society, pp. 539–543, 1999.*