Modelos Dos Actores para grupos de procesos

Karina M. Cenci * Jorge R. Ardenghi **

Laboratorio de Investigación en Sistemas Distribuidos
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Resumen

In distributed systems, applications compete in the use of a resource or work together and need a resource. To keep these requirements it is necessary a protocol to guarantee the access to the shared resources. In this article, we introduce a model like a pattern to solve the problem mutual exclusion of group of processes. This model contains two actors, groups and processes, which interact among themselves. We present some examples of solutions applying this model, using message and shared memory to communicate.

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

Resumen

En sistemas distribuidos las aplicaciones realizan trabajos que requieren acceso en forma exclusiva a un recurso o realizan trabajo en forma conjunta para el cual requieren compartir la utilización de un recurso, para poder mantener estos requerimientos es necesario contar con protocolos que garanticen ambas propiedades. En este trabajo, se presenta un modelo como patrón para resolver el problema de exclusión mutua para grupo de procesos. El modelo está compuesto por dos actores, grupos y procesos, que interactúan entre sí. Se presentan un par de soluciones utilizando este modelo, sobre memoria compartida y pasaje de mensajes.

Palabras Claves: Sistemas Distribuidos - Exclusión Mutua - Exclusión Mutua para Grupos de Procesos - Concurrencia -

*e-mail: kmc@cs.uns.edu.ar
**e-mail: jra@cs.uns.edu.ar

1. Introducción

Desde el comienzo de los sistemas con multiprogramación fue necesario disponer de un protocolo que garantice el uso exclusivo de un recurso, este problema es conocido como *exclusión mutua*. Con el avance tecnológico, el surgimiento y crecimiento de los sistemas distribuidos, algunas de las aplicaciones realizan trabajo cooperativo y requieren exclusión mutua sobre un grupo de procesos y con otro grupo de procesos compartir el/los recursos para resolver su tarea.

En la literatura, para el problema de *exclusión mutua para grupos de procesos*, hay una variedad de propuestas [8], [9], [16], [4], [5], [6], las mismas utilizan diferentes modelos de implementación. Este problema es una extensión del problema original, donde cada proceso puede compartir con otros procesos la sección crítica (utilización del recurso) si es necesario para resolver su tarea. En estas propuestas se garantizan las propiedades de un buen algoritmo de exclusión mutua incorporándole como nueva propiedad la concurrencia de procesos cooperativos.

Para el problema original de exclusión mutua se presentaron una gran variedad de soluciones [1], [2], [3], [11], [13], [14], [15], las mismas están basadas en el modelo de memoria compartida distribuida y otros en pasaje de mensajes. En este trabajo se presenta un modelo basado en dos actores para resolver el problema de Exclusión Mutua para Grupos de Procesos, que se puede aplicar a los algoritmos que resuelven el problema de exclusión mutua original. Se muestran implementaciones de este modelo aplicadas a algoritmos basados en memoria compartida y en pasaje de mensajes.

2. Preliminares

Se considera un conjunto de n procesos $p_0, p_1, .., p_{n-1}$ los cuales trabajan en forma independiente o en forma cooperativa en un grupo.

Los procesos pueden participar de cualquiera de los diferentes m grupos G_0, G_1, ... , G_{m-1}. Cada uno de los grupos, cuando se encuentra activo utiliza el/los recurso/s por los cuales compite en el sistema. En un determinado instante, sólo un grupo puede estar *activo*. En el grupo, participan todos los procesos que estén interesados en el trabajo.

Inicialmente cada uno de los procesos está trabajando individualmente. Cuando desea trabajar en equipo, elige el *grupo*. Se considera que cada proceso trabaja en equipo por un tiempo finito, y que puede participar en cualquiera de los diferentes grupos. En la figura 1, se observa que los procesos P_1, P_2 y P_7 están actualmente vinculados al grupo G_3; éste se encuentra activo y con permiso para utilizar el recurso, significando que los procesos que lo integran están concurrentemente utilizando el recurso. Los procesos P_0 y P_8 están integrando el grupo G_1 que está compitiendo por alcanzar el permiso de utilizar el recurso.

En el caso, en que se considerara que cada grupo estuviera formado por un solo proceso, entonces el problema se reduciría al modelo convencional de exclusión mutua para n procesos, donde solamente un proceso a la vez puede estar en la sección crítica. Para resolver este problema se requiere una extensión del problema de la exclusión mutua al caso donde k procesos pueden compartir la utilización del recurso en un instante de tiempo. Se requiere un algoritmo que satisfaga los siguientes requerimientos:

- *Exclusión Mutua*: si algún proceso está trabajando en un grupo, no puede haber otro proceso trabajando en un grupo diferente simultáneamente.

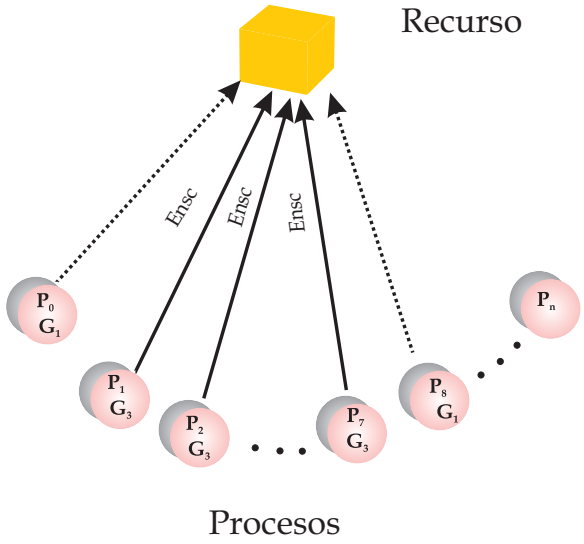

Figura 1: Relación entre grupo y procesos

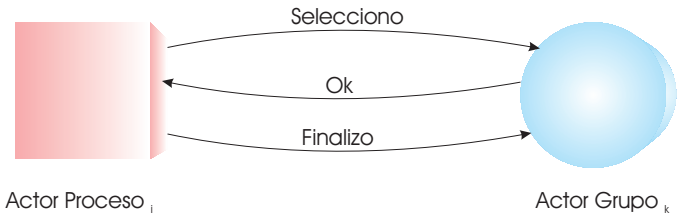

Figura 2: Comunicación entre los 2 Actores

- *Demora Limitada (libre de inanición)*: un proceso que desea participar de un grupo eventualmente tendrá éxito.
- *Entrada Concurrente*: si algunos procesos están interesados en un grupo y no hay un proceso interesado en otro grupo, entonces los procesos pueden participar concurrentemente del grupo.
- *Libre de interbloqueo*: cuando la sección crítica está disponible, los grupos no deberían esperar indefinidamente y alguno debería obtener el permiso para acceder.

3. Presentación del Modelo de dos actores

La propuesta de este modelo está compuesta por 2 tipos de actores: los *procesos* y los *grupos*, que se integran para competir por la utilización de un recurso. El *actor proceso* selecciona un grupo de trabajo, y el *actor grupo* compite para acceder a la sección crítica. En la figura 2 se observa la relación entre los 2 componentes y en la figura 3 se muestra un ejemplo de competición y concurrencia.

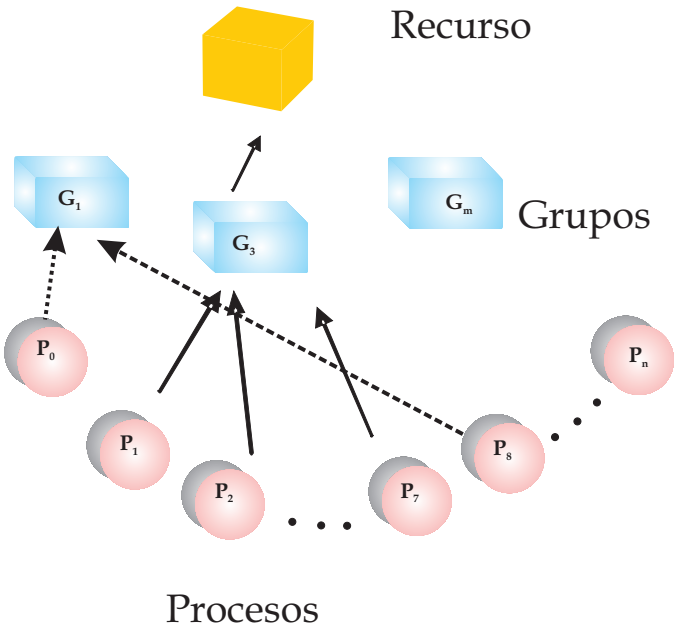

Figura 3: Ejemplo de Competición y Concurrencia - 2 Actores

Cuando un actor no está involucrado de ninguna manera con el recurso, se dice que está en la sección *resto*. Para obtener la admisión a la sección crítica, el actor ejecuta un protocolo de entrada (*trying*), después que utiliza el recurso, se ejecuta un protocolo de salida (*exit*). Este procedimiento puede repetirse, de modo que cada actor sigue un ciclo, desplazándose desde la *sección resto*, a la *sección de entrada*, luego a la *sección crítica (SC)* y por último a la *sección de salida*, y luego vuelve a comenzar el ciclo en la *sección resto*.

Como se observa en la figura 4, el primer paso que realiza el *actor proceso*, en la sección de entrada, es seleccionar el grupo en el cual desea participar del conjunto de m grupos. El segundo paso es esperar hasta que el grupo seleccionado entre en la sección crítica para que pueda acceder a la misma. Cuando finaliza su actividad, sale de la sección crítica, se desvincula del grupo (sección de salida). En la figura 5 se muestran los estados del mismo.

El *actor grupo* inicialmente está inactivo, en la sección resto (sección inactivo), esto representa que ningún proceso lo ha seleccionado para participar en el mismo. El primer proceso que lo selecciona para participar, hace que comience la competencia por entrar en la sección crítica, y se lo identifica como el primer proceso que pertenece al grupo; pasa a la sección de entrada. Todos los procesos que lo seleccionen mientras se encuentra en competición por entrar a la sección crítica, se agregan a los procesos ya existentes. En el caso que el grupo esté en la sección crítica, si el proceso que activó al grupo está trabajando en la misma, entonces el proceso se incorpora, sino se pone en cola de espera hasta que termine la actual vuelta (todos los procesos que están trabajando finalicen su tarea y el grupo salga de la sección crítica), se reinicie el ciclo, esto es, compita nuevamente por el ingreso en la sección crítica. En la figura 6 se observa el comportamiento del actor grupo mientras se encuentra activo y en la figura 7 se muestran los estados del mismo.

Proceso$_i$

1. ..... {Sección Resto}
2. El proceso selecciona el grupo de trabajo {Grupo$_k$}
3. Espera hasta que entra a la sección crítica
   {Sección de Entrada}
4. ..... { Sección Crítica}
5. Sale de la sección crítica y se desvincula del grupo.
6. ..... {Sección Resto}

Figura 4: *Esquema del Actor Proceso*

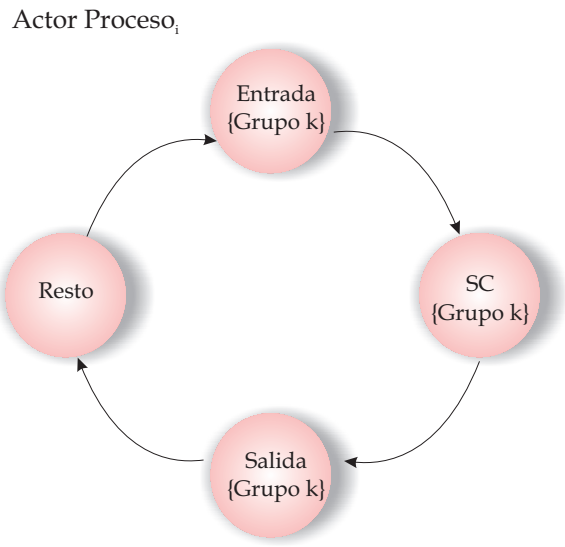

Figura 5: Estados del Actor Proceso

Grupo$_k$

1. ..... {Sección Resto (Inactivo) }
2. Un proceso lo selecciona para trabajar en él.
   {Sección de Entrada (Compitiendo)}
3. Si es el primer proceso en el grupo entonces
   Comienza a competir en el ingreso a la S.C.
4. sino
   Si no está en la S.C. entonces
   Agrega el proceso a la lista de procesos
   sino
   Si está el primero del Grupo entonces
   entra en la S.C. el proceso
   sino
   El proceso lo coloca en la lista de espera hasta que termine la sección crítica actual y compita por ingresar nuevamente
5. Si no hay ningún proceso en la Sección Crítica libera el recurso.
6. ..... {Sección Resto (Inactivo)}

Figura 6: *Esquema del Actor Grupo*

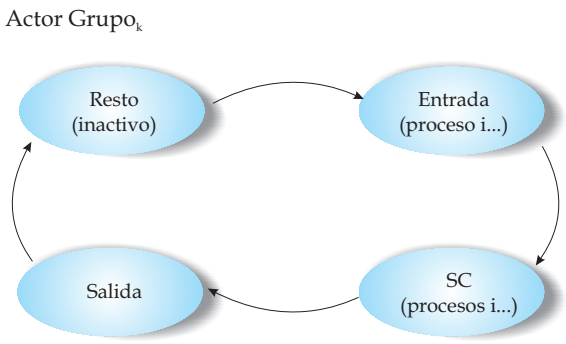

Figura 7: Estados del Actor Grupo

Los *actores grupos* compiten por alcanzar el permiso para acceder al recurso (acceso a la sección crítica), y sólo un único grupo tiene derecho de utilizar el recurso en un determinado instante de tiempo.

4. Aplicación del Modelo

En esta sección, se brindan un par de algoritmos de *exclusión mutua para grupos de procesos*, basado en memoria compartida distribuida y pasaje de mensajes, que aplican el modelo presentado. También se han desarrollado otros algoritmos aplicando este modelo para algoritmos que sean adaptivos [6], con restricciones de tiempo [7].

4.1 Memoria Compartida

Este algoritmo fue presentado en [4], utiliza el paradigma de memoria compartida distribuida sobre las variables de control utilizadas. En la figura 8 se muestran las variables compartidas requeridas para la resolución del problema.

Variables Compartidas
$\forall$ flag(i): $0 \leq i \leq (m\text{-}1)$, flag(i) = 0 inicialmente
para cada cadena binaria x de a lo sumo longitud etapas-1

turn(x) inicialmente arbitraria, escrito y leído por exactamente aquellos grupos i para los cuales x es un prefijo de la representación binaria de i.

$\forall$ lista(i,j): $0 \leq i \leq (m\text{-}1)$, $0 \leq j \leq (n\text{-}1)$,
lista(i,j) = <0, resto> inicialmente

etapas = (Si truncar(log(m)) = log(m) entonces = truncar(log(m)) sino = truncar(log(m)) + 1 fin si)

Figura 8: Variables Compartidas

Características de las variables compartidas:

- *flag(i)* es escrita por el $grupo_i$, leída por el resto de los grupos y los procesos que se asocian al mismo, especifica el nivel de competencia.
- *lista* contiene los procesos que están asociados en un instante de tiempo a los grupos.
- *lista(i,j)* es escrita por el $proceso_j$ que se asocia al $grupo_i$ y leída por el $grupo_i$ y todos los procesos que se asocian al mismo grupo.
- *Etapas* contiene el número de niveles de competencia.

En la figura 9, se observa el comportamiento del *actor grupo*. Inicialmente, está esperando que algún proceso lo seleccione, inicializando la variable *lista(i,j).estado* a "espera". Cuando ocurre este evento, el grupo se activa y comienza su competencia por ingresar en la sección crítica. Indica a los procesos asociados al mismo que alcanzó el objetivo inicializando *flag(i)* a etapas + 1. Permanece en la sección crítica mientras haya procesos activos en el grupo. Cuando todos los procesos salen de la sección crítica, el grupo libera el recurso, inicializando el *flag(i)* a 0 y comienza otra vez el ciclo, esperando por un nuevo proceso que lo seleccione.

Grupo$_i$
Entrada$_i$
waitfor [∃ j: 1..n, lista[k,j] = < .., espera>]
Bucar_lider(lista,i)
para k = 1 hasta etapas hacer

flag(i) = k {representa los diferentes niveles}

Si (role(i,k)≠0) ó (i≤m-k) entonces

Waitfor [∀ j ∈ oponentes(i,k) : flag(j) < k] ó [turn(comp(i,k))≠role(i,k)]

flag(i)= etapas + 1 {para que el proceso sepa que está en la S.C.}
... *Sección Crítica*

Waitfor [∀ j: 1..n, lista(i,j)≠<..,en_cs>]

flag(i) = 0

Figura 9: Comportamiento Actor Grupo

Proceso$_i$
... *Región Resto*
Entrada$_i$
Selección del grupo en g
Si inactivo(g) entonces

lista(g,i) = <2, espera> {Es el primer proceso en el grupo, habilita mientras está en la sección crítica que otro procesos puedan participar concurrentemente en la misma}

sino

lista(g,i) = <1, espera> {Por lo menos hay otro proceso que estaba en el grupo}

fin si
Waitfor (flag(g) = etapas + 1) ∧ ((lista(g,i)=<2, espera>) ∨ (∃ j: 1..n, lista(g,i)=<2, en_cs>))
lista(g,i)=<.., en_cs>
... *Sección Crítica*
Salida$_i$
lista(g,i) = <0, resto>

inactivo(g) ≡ (flag(g) = 0) {Indica que el grupo está en la región resto}

Figura 10: Comportamiento Actor Proceso

El algoritmo cumple con las condiciones de *buena formación, exclusión mutua y progreso* que requiere para resolver el problema de la exclusión mutua; y además satisface los requisitos de un buen algoritmo de exclusión mutua, esto es, *Libre de Interbloqueo*,*Libre de inanición* e *Imparcialidad*.

Los procesos trabajan en forma individual y cuando desean trabajar en equipo seleccionan el grupo en el cual quieren participar. En la figura 10, se muestra el comportamiento que realiza cada uno de los *actores procesos*, verifican si el grupo está inactivo al momento de la selección, si es así se considera que es el primer proceso del grupo.

Para garantizar que un grupo no permanezca indefinidamente en la región crítica y se alcance un alto grado de concurrencia entre los procesos que trabajan cooperativamente, el proceso al verificar el nivel de competencia del grupo, también controla si el primer proceso (el líder) del grupo está actualmente trabajando en la región crítica. Si el proceso líder no está activo entonces espera la otra vuelta.

Si un proceso selecciona un grupo que tiene el acceso a la sección crítica pueden suceder los siguientes casos:

1. El primer proceso del grupo se encuentra activo, entonces habilita al nuevo proceso a que trabaje en el grupo cooperativamente, sin tener que esperar. 2. El primer proceso del grupo no se encuentra activo, entonces el proceso se agrega a la lista de procesos en espera. Debe esperar que termine el grupo la utilización de la sección crítica y vuelva a competir por el recurso, en el caso que todos los grupos estén compitiendo por acceder a la sección, debería esperar en el O(m+etapas) ciclos para acceder nuevamente a la misma, esto es esperar que entren los grupos restantes.

4.2 Algoritmo de Exclusión Mutua para Grupos

Este algoritmo fue presentado en [5], su implementación está basada en pasaje de mensajes utilizando quorum para acceder a la sección crítica. Se considera que se tienen P_1, ..., P_M procesos y G_1, ..., G_N grupos. Este algoritmo supone que la red es confiable y no necesita reconocimiento.

El modelo está formado por dos actores que interactúan que son los *actores procesos* y los *actores grupos*. El *actor proceso* es un componente *activo*, ya que es el que selecciona el grupo para acceder al recurso.

El *actor grupo* es un componente *pasivo*, ya que está esperando que un proceso lo seleccione para participar del mismo, cuando esto ocurre entonces empieza a competir por el recurso.

Actor Proceso

Cada actor proceso i, $1 \leq i \leq M$, realiza los siguientes pasos: selecciona el grupo en el cual va a trabajar, le envía un mensaje de solicitud de requerimiento y espera hasta que su requerimiento es aceptado, luego ingresa a la sección crítica por un tiempo limitado, y al salir le comunica al grupo que finalizó su trabajo en la sección crítica. Cada proceso está vinculado al grupo mientras utiliza el *recurso* y luego se desvincula del mismo; podría vincularse con diferentes grupos a través del tiempo. Se considera que un proceso está vinculado a un grupo un tiempo finito, esto es, para garantizar la propiedad de libre de inanición.

Los mensajes que realiza el actor proceso son los siguientes:

- Req_Proceso(G_k, P_i): el proceso P_i envía un mensaje de solicitud para participar en el grupo G_k y utilizar el recurso.
- Rep_Proceso(G_k, P_i): el proceso P_i recibe la respuesta a la solicitud realizada al grupo G_k y puede acceder al recurso.
- Rep_Proceso_Fin(G_k, P_i): el proceso P_i le indica al grupo G_k que ha finalizado su participación del grupo y que se desvincula del mismo.

Actor Grupo

El actor grupo está inactivo hasta que recibe un miembro que temporalmente va a trabajar en él. Cuando recibe un mensaje de *Req_Proceso* entonces el grupo inicia su competencia por acceder a la sección crítica. Un único grupo puede utilizar el recurso en un instante de tiempo, pero en ese

instante de tiempo pueden compartir el trabajo varios procesos. La competencia la inicia enviando un mensaje *multicast* a todos los miembros de su quorum, y espera que todos le envíen el *lock* para acceder a la exclusión mutua.

El actor *grupo* puede recibir un conjunto de mensajes y de acuerdo al requerimiento su estado actual puede modificarse y además generar nuevos mensajes a otros actores. Utiliza un conjunto de variables locales que mantienen su estado, estas son:

estado: mantiene el estado actual del grupo, los valores que pueden contener son RESTO (INACTIVO), COMPITIENDO (ENTRADA), SC y SALIDA.

LP: mantiene información de todos los procesos que quieren utilizar el grupo.

LG: mantiene información de todas las solicitudes de *lock* que están pendientes.

lider: mantiene identificado al proceso que activó el grupo mientras el mismo pertenezca al grupo.

Cada uno de los mensajes que recibe se los puede clasificar como proveniente de un actor *proceso* o de un actor *grupo*. Los mensajes afectan el estado del *grupo*, a continuación se presentan para cada tipo de mensaje las acciones que involucran.

♯ Req_Proceso(G_k, P_i): proviene de un proceso P_i, si el grupo está esperando que lo seleccione un proceso, esto es, su estado es RESTO (INACTIVO) entonces lo activa y el proceso P_i se convierte en el proceso líder del grupo. Si el grupo está en el estado COMPITIENDO (ENTRADA) se agrega el proceso a la Lista de Procesos. Si el grupo está en SC (en *Sección Crítica*) y el líder está activo entonces le avisa al proceso que puede comenzar a trabajar sino lo agrega a la Lista de Procesos (LG).

♯ Req_Proceso_Fin(G_k, P_i): proviene de un proceso P_i que avisa al grupo que ha finalizado su trabajo en el mismo. Si el proceso es el líder entonces lo deshabilita. Si es el último proceso libera la sección crítica y vuelve al estado RESTO(INACTIVO) y controla si hay procesos que están esperando para ingresar al grupo para iniciar nuevamente la competencia por alcanzar la sección crítica.

♯ Req_Grupo(G_l, priori): proviene del grupo G_l que requiere el *lock* del grupo G_k. El grupo G_k otorgará el *lock* si lo tiene disponible. Sino tiene disponible el *lock* pueden ocurrir dos casos diferentes: (a) La prioridad del mensaje recibido es menor que la prioridad del mensaje cedido el *lock* entonces el requerimiento es demorado. (b) Si la prioridad es mayor entonces reclamará el *lock* al grupo correspondiente para luego cederla al de mayor prioridad.

♯ Rec_Grupo(G_l, G_k): proviene del grupo G_l, como respuesta afirmativa al mensaje Req_Grupo de requerimiento de lock. Si el grupo G_k tiene todos los *locks* entonces cambia su estado a SC y avisa a todos los procesos que están asociados al grupo.

♯ Rel_Grupo(G_l, G_k): proviene del grupo G_l solicitando que le devuelva el *lock*, esto será exitoso en el caso que el grupo G_k no esté en la sección crítica.

♯ Rep_Rel_Grupo(G_l, G_k): proviene del grupo G_l liberando el *lock* que había dado el grupo G_k. El *lock* es cedido al requerimiento con mayor prioridad.

♯ Lib_Grupo(G_l): proviene del grupo G_l comunicando que terminó su tiempo en la sección crítica. Si hay requerimientos pendientes entonces elige el de mayor prioridad y le otorga el *lock*

El algoritmo garantiza las propiedades de un buen algoritmo.Cuando un proceso quiere trabajar en un grupo y es el líder del mismo, entonces se requieren 3 + 3(k-1) mensajes para que pueda acceder a la sección crítica. Si en un grupo están trabajando en forma concurrente x procesos entonces para los x-1 procesos se requiere 3(x-1) mensajes para acceder a la sección crítica y en total $3x$ + 3(k-1) mensajes. En el caso en el cual se deban devolver todos los locks por solicitudes de requerimiento de mayor prioridad, se agregan k-1 mensajes de entrega del lock y otros k-1 mensajes de otorgamiento del lock. Si el grupo tuviera l procesos trabajando en forma concurrente entonces requerirían $3x$ + 5(k-1) mensajes.

5. Conclusión

Para el problema tradicional de exclusión mutua, existen una gran variedad de soluciones. Estas se pueden clasificar de acuerdo al paradigma de implementación en pasaje de mensajes ó memoria compartida distribuida; también se los puede clasificar de acuerdo a la propiedad que priorizan: minimizar la cantidad de accesos en el mejor caso, acotar la cantidad de accesos en el peor caso, acotar el tiempo de espera para ingresar, la adaptabilidad, las restricciones de tiempo, cantidad de mensajes requeridos, etc. El modelo de 2 actores presentado, permite obtener algoritmos que resuelvan el problema de exclusión mutua para un grupo de procesos a partir de los algoritmos que resuelven el problema tradicional de exclusión mutua. Utilizando el mismo, se puede obtener un algoritmo que resuelva el problema de exclusión mutua para grupos de procesos que mejor se adapte a la situación a resolver. Se muestran ejemplos de algoritmos obtenidos a partir de este modelo propuesto, donde se utilizan los diferentes paradigmas de implementación.

Referencias

[1] D. Barbara, H. García-Molina. Mutual exclusion in partitioned distributed systems. *Distributed Computing*, vol. 1, no. 2, pp. 119–132, 1986.

[2] J. E. Burns, P. Jackson, N. A. Lynch, M. J. Fischer, G. L. Peterson. Data Requirements for Implementation of N-Process Mutual Exclusion Using a Single Shared Variable *Journal of the ACM*, vol. 29, issue 1, 1982.

[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 *Modelo Asincrónico Adaptativo de Exclusión para Grupos de Procesos* CACIC 2005.

[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 PODC.

[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] L. Lamport. A Fast Mutual Exclusion Algorithm. *ACM on Transactions on Computer Systems*, vol. 5, no. 1, Febrero 1987.

[12] Nancy A Lynch. *Distributed Algorithms*, 1997.

[13] 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.

[14] Michael Raynal. *Algorithms for Mutual Exclusion*. MIT Press, Cambridge, 1986.

[15] Gary L. Peterson, Myths about the mutual exclusion problem. *Information Processing Letters*, Junio 1981.

[16] K. Vidyasankar. A simple group mutual l-exclusion algorithm. *Information Processing Letters*, vol. 85, no. 2, pp. 79–85, Enero 2003.

[17] 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.