Exclusión Mutua para Grupos de Procesos utilizando un actor

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

Las aplicaciones distribuidas están formadas por un conjunto de procesos, los cuales pueden competir por utilizar un recurso o trabajar en forma conjunta para resolver una tarea.

Estas aplicaciones requieren protocolos que permitan concurrencia entre los procesos que trabajan cooperativamente y exclusión mutua para aquellos que compiten por utilizar el recurso. En este trabajo se presenta un algoritmo para exclusión mutua para grupos de procesos basándose en el modelo de memoria compartida asincrónica, donde cada conjunto de procesos que realice una actividad conjunta compiten conjuntamente para acceder al recurso.
...

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

Las aplicaciones utilizan recursos, algunas requieren uso exclusivo de los mismo y otras que realizan trabajo cooperativo comparten el acceso a esos recursos. A partir de estos requerimientos, es necesario considerar en las aplicaciones distribuidas protocolos que soporten las características de *exclusión mutua* y *concurrencia*.

- Exclusión Mutua: garantiza el acceso exclusivo a un recurso común sobre un conjunto de procesos compitiendo.
- Concurrencia: permite que los procesos no conflictivos compartan un recurso para incrementar la performance del sistema.

La característica de exclusión mutua es el primer problema que surge en los sistemas multiprogramados, en la literatura hay muchos protocolos propuestos que garantizan esta propiedad, como por ejemplo en [2], [3], [4], [7], [10], ..., algunos están basado en el modelo de memoria compartida distribuida y otros en pasaje de mensajes.

Para considerar la situación que varios procesos comparten el acceso a un recurso, se extiende el problema original de *exclusión mutua* a *exclusión mutua para grupos de procesos*. En este trabajo se consideran protocolos para la extensión a grupos de procesos basándose en el modelo de memoria compartida, dónde los procesos se comunican mediante la escritura y lectura de variables compartidas.

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 una actividad que utiliza un recurso compartido. En el resto del trabajo se considera que la actividad compartida se realiza en un grupo con el mismo interés.

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 compiten en el sistema. En un determinado instante, sólo un grupo puede estar utilizando el recurso compartido. En el grupo, participan todos los procesos que están interesados en la actividad.

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 observan 2 formas diferentes de modelar el mismo problema. En el caso (A), los procesos P_1, P_2 y P_7 que están actualmente vinculados al grupo G_3; y los procesos P_0 y P_8 están integrando el grupo G_1. La competencia para acceder al recurso la ganó el grupo G_3, el mismo se encuentra en la sección crítica y todos los procesos vinculados están trajando cooperativamente, el grupo G_1 está compitiendo por alcanzar el permiso de utilizar el recurso. En el caso (B), los procesos P_1, P_2 y P_7 eligieron al grupo G_3 para trabajar concurrentemente, y los procesos P_0 y P_8 eligieron al grupo G_1 para trabajar concurrentemente. Cada proceso compitió por acceder al recurso, uno de los procesos que eligió al grupo G_3 obtiene el permiso entonces permite que accedan los otros procesos que quieren realizar la misma actividad cooperativamente.

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

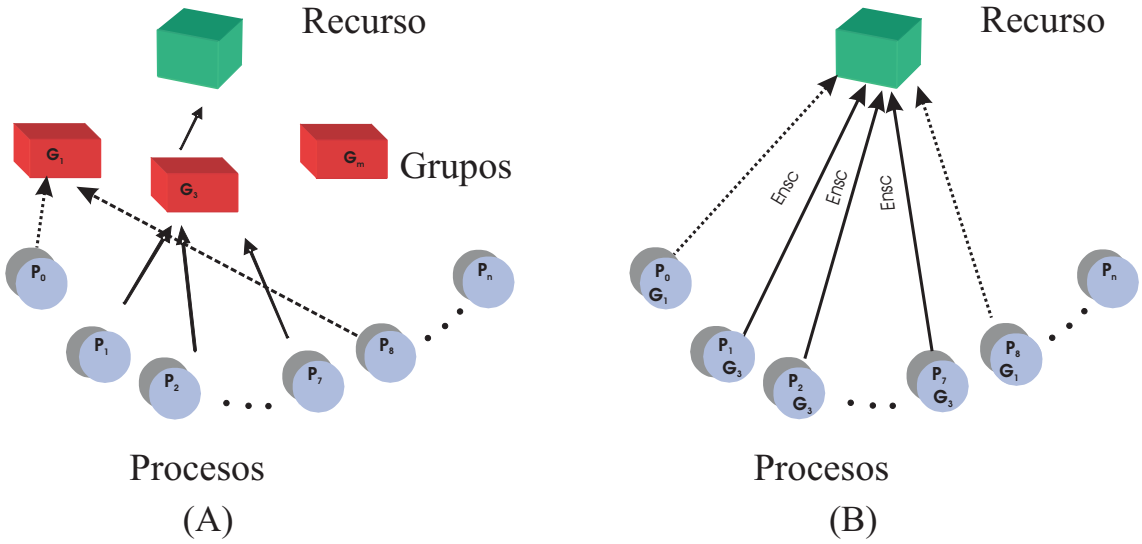

Figura 1: Ejemplo de Concurrencia y Competición

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 simultánea-mente.
- *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.

2. Algoritmo con dos actores

El algoritmo presentado en [5] resuelve el problema de exclusión mutua para grupos de procesos utilizando dos tipos de actores: los *procesos* y los *grupos*, que se integran para competir por la utilización de un recurso. En el modelo, se tienen dos componentes que formarán parte del algoritmo, el proceso que selecciona un grupo de trabajo, y el grupo que compite para acceder a la sección crítica.

Cuando un actor no está involucrado de ninguna manera con el recurso, se dice que está en la región *resto*. Para obtener la admisión a la región crítica, un 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 *región resto (R)*,

```
Proceso_i
... Sección 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 2: Actor Proceso

```
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

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

    flag(i) = 0
```

Figura 3: Actor Grupo

a la *región de entrada (T)*, luego a la *región crítica (C)* y por último a la *región de salida (E)*, y luego vuelve a comenzar el ciclo en la *región resto*.

Como se observa en la figura 2, el primer paso que realiza el *actor proceso*, en la región 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 región crítica para que pueda acceder a la misma. Cuando finaliza su actividad, sale de la región crítica, se desvincula del grupo (región de salida).

Como se observa en la figura 3, el *actor grupo* inicialmente está inactivo, en la región resto, 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 región crítica, y se lo identifica como el primer proceso que pertenece al grupo; pasa a la región de entrada. Todos los procesos que lo seleccionen mientras se encuentra en competición por entrar a la región crítica, se agregan a los procesos ya existentes. En el caso que el grupo esté en la región 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 región crítica), se reinicie el ciclo, esto es, compita nuevamente por el ingreso en la región crítica.

Los *actores grupos* compiten por alcanzar el permiso de utilizar el recurso (acceso a la región

∀ flag(i): $0 \leq i \leq (m-1)$, 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.

∀ lista(i,j): $0 \leq i \leq (m-1)$, $0 \leq j \leq (n-1)$,
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 4: Algoritmo dos actores - variables compartidas

Notas

• comp(k,l)→ el nivel l del grupo k, es la cadena que consiste de los (etapas-l) bits de mayor orden de la representación binaria de k.

• etapas → está representando la cantidad de bits necesaria para almacenar hasta el valor (m-1)

• role(k,l) → el rol del $grupo_k$ en el nivel l de competeción del mismo, es el bit (etapas-l+1) de la representación binaria de k (representa si desciende de la rama derecha o izquierda)

• oponentes(k,l) → los oponentes del $grupo_k$ en el nivel l de competición, es el conjunto de índices de grupos con el mismo orden de bits en (etapas-l) y el opuesto (etapas-l+1)

crítica), sólo un único grupo tiene derecho de utilizar el recurso en un determinado instante de tiempo. El esquema base para la competencia de los grupos, está basado en el algoritmo de Tournament, con la extensión a m elementos, donde m no es necesariamente una potencia de 2. Las variables utilizadas se muestran en la figura 4.

Cada grupo está ocupado en una serie de competencias de O(log n) para obtener el recurso. Puede considerarse que la competición está dispuesta en un árbol de competencia binario. Las hojas corresponden a los m grupos. En las Notas se presentan las funciones que se utilizan en el algoritmo.

La idea es que el algoritmo cumpla las siguientes condiciones:

- Un único grupo está activo utilizando el recurso compartido (exclusión mutua)
- Si el recurso está disponible y un grupo quiere utilizarlo (está en espera) que acceda al mismo sin tener más demora.
- Si un grupo está activo y el primer proceso también, todo proceso que quiera trabajar en el mismo que lo pueda realizar, de esa manera se logra un mayor nivel de concurrencia.

El algoritmo cumple con las condiciones de *buena formación, exclusión mutua y progreso* que requiere para resolver el problema de la exclusión mutua (porque está basado en el algoritmo

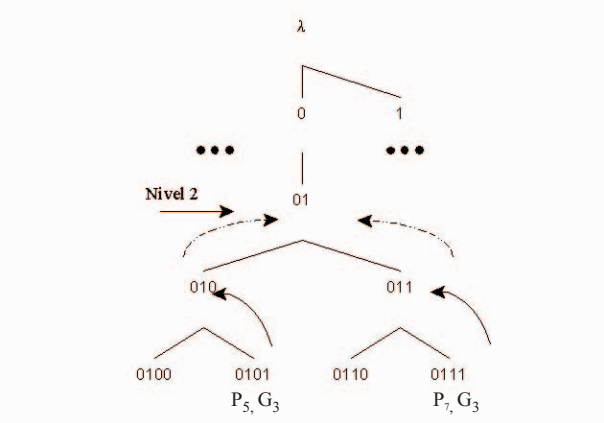

Figura 5: Procesos Concurrentes

presentado en [4]); y además satisface los requisitos de un buen algoritmo de exclusión mutua, esto es, *Libre de Interbloqueo*, *Libre de inanición* e *Imparcialidad*.

3. Presentación del Modelo

El modelo presentado está formado por un actor *proceso* que utiliza el recurso en forma compartida con otros actores *procesos* que realizan su trabajo en forma conjunta, esto es participan del mismo grupo de interés.

El actor *proceso* es el que compite por acceder al recurso participando de un determinado *grupo*. El actor *proceso* puede participar de diferentes grupos durante en el transcurso de su trabajo, pero en un determinado instante de tiempo participa de un único grupo.

En el caso (B) de la figura 1, se puede observar el comportamiento del modelo presentado. El esquema de competencia del actor *proceso* está basado en el algoritmo de Tournament, con la extensión a n elementos, donde n no es necesariamente una potencia de 2.

El proceso inicialmente está en la sección resto, cuando ingresa en la sección de entrada selecciona el grupo en el cual va a participar y comienza la competencia por acceder al recurso. La competencia se realiza por niveles, cuando supera el último nivel puede acceder a la sección crítica, si en un determinado nivel se encuentra con un proceso que quiere acceder por el mismo grupo entonces compiten conjuntamente para acceder a la sección crítica. En la figura 5, se observa la situación en la cual los procesos P_5 y P_7 que compiten en el nivel 2 comparten el mismo grupo G_3, entonces avanzan juntos.

En el modelo presentado cuando un proceso P_i compite con el proceso P_j en el nivel k identifica que es un compañero entonces espera hasta que el proceso P_j acceda al recurso y luego participar en forma cooperativa.

4. Algoritmo con un actor

El algoritmo presentado se basa en el paradigma de memoria compartida distribuida sobre las variables de control utilizadas. En la figura 6 se muestran las variables compartidas.

∀ flag(i): 0 ≤ i ≤ (n-1), flag(i) = 0 inicialmente
∀ grupo(i): 0 ≤ i ≤ (n-1), grupo(i) = -1 inicialmente
∀ oponente_compa(i,i): 0 ≤ i ≤ (n-1), oponente_compa(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.

etapas = (Si truncar(log(n)) = log(n) entonces = truncar(log(n)) sino = truncar(log(n)) + 1 fin si)
pganador = -1

gganador = -1

Figura 6: Variables Compartidas

Las variables *flag(i)* y *grupo(i)* son escritas por el proceso P_i y leídas por el resto de los procesos; la primer variable indica el nivel de competencia del proceso y la segunda en cuál grupo está vinculada. Las variables *pganador* y *gganador* pueden ser leídas y escritas por todos los procesos; la variable *pganador* contiene la información de cuál es el primer proceso que ingresó en la sección crítica de todos los que trabajan cooperativamente y *gganandor* contiene la información de cuál es el grupo al que pertenecen los procesos que están en la sección crítica. *Etapas* contiene el número de niveles de competencia. Las variables $oponente\_compa(0..n-1, i)$ son escritas por el proceso P_i y leídas por el proceso P_j.

En la figura 7, se muestra el comportamiento del protocolo de entrada y salida en formato tradicional. Se considera los accesos a las variables compartidas atómicos. Un buen algoritmo de exclusión mutua garantiza las condiciones de buena formación, exclusión mutua y progreso. El modelo está basado en [4] que garantiza las condiciones de un buen algoritmo, se le incorporó el avance de procesos oponentes que son compañeros para acceder conjuntamente a sección crítica. En el caso que todos los oponentes sean competidores el algoritmo garantiza todas las condiciones. ¿Qué sucede si un proceso P_i en el nivel k obtiene que un oponente P_j es compañero? El estado del proceso P_i sería el siguiente:

- $flag(i) = \text{k}$
- $oponente\_compa(j, i) = 1$
- $grupo(i) = \text{g}$

El proceso P_j podría estar en las siguientes situaciones:

Nivel k	Nivel Superior	En Sección Crítica
$flag(j) = \text{k}$	$flag(j) > \text{k}$ y $<$ etapas + 1	$flag(j) =$ etapas + 1
$grupo(j) = \text{g}$	$grupo(j) = \text{g}$	$grupo(j) = \text{g}$

Si está en el mismo nivel entonces avanza al próximo nivel de competición. Si está en un nivel superior o en la sección crítica lo que modifica es cuando se encuentre en la sección de salida, ya que debe esperar a que el proceso P_i tome una decisión, acceda a la sección crítica o haya perdido el permiso de acceder directamente y tenga que continuar compitiendo.

Proceso$_i$

... *Sección Resto*

Entrada$_i$
g = seleccionar grupo
grupo(i) = g
k = 1
mientras (k $\leq$ etapas) hacer {
flag(i) = k
turn(comp(i,k)) = role(i,k)
si (role(i,k) $\neq$ 0) ó (i $\leq$ n-k) ó (i $< 2^{etapas}$ / 2) entonces
waitfor [$\forall j \in$ Oponentes(i,k) : flag(j) $<$ k] ó [Hay_Compa(i,k)] ó [turn(comp(i,k) $\neq$ role(i,k)]
Si Hay_Compa(i,k) entonces
oponente_compa(j,i) = 1
waitfor [ganar(i)] ó [$\nexists$ Oponente(i,k) : (flag(j) $\geq$ k) $\wedge$ (grupo(j) == grupo(i))]
Si ganar(i) entonces k = etapas, flag(i) = etapas + 1
oponente_compa(j,i) = 0
k = k + 1}
flag(i) = etapas + 1
Si (pganador == -1) entonces
pganador = i
gganador = grupo(i)

... *Sección Crítica*

Salida$_i$
Si (pganador == i) entonces
pganador = -1
gganador = -1
waitfor ($\forall$ j, 0 $\leq$ j $\leq$ (n-1) oponente_compa(i,j) == 0)
grupo(i) = -1
flag(i) = 0

Hay_Compa(i,k) $\equiv$ *Si*($\exists j \in$ Oponentes(i,k) : flag(j) $\geq$ k $\wedge$ grupo(j) == grupo(i)) Entonces Verdadero Sino Falso

ganar(i) $\equiv$ Si (gganador == grupo(i) $\wedge$ pganador $\neq$ -1) Entonces Verdadero Sino Falso

Figura 7: Algoritmo de un actor

Para garantizar la equidad, una vez que el primer proceso que ingresó a la sección crítica en un grupo G_l, finaliza su sección crítica no se permite que ingresen nuevos procesos vinculados al grupo G_l.

5. Medidas

Las cuestiones a tener en cuenta para obtener las medidas en la complejidad de tiempo son:

- Complejidad en un paso remoto (referencias de memoria remota) de un algoritmo es el número máximo de operaciones de memoria compartida requeridas por un proceso para ingresar y salir de su sección crítica, asumiendo que cada sentencia await es contabilizada como un única operación.
- El tiempo de respuesta del sistema es el intervalo de tiempo entre entradas a la sección crítica.

Otro factor importante para determinar la velocidad de un algoritmo es la cantidad de tráfico de interconexión que el genera. En función de este otro parámetro se define a la complejidad de tiempo de un algoritmo de exclusión mutua a ser el peor caso en el número de referencias de memoria remota por un proceso en orden para ingresar y salir de su sección crítica.

En el algoritmo presentado, cada proceso P_i compite en diferentes niveles, la cantidad máxima de niveles es del O(log(n)). En el caso que los n procesos quieran acceder a la sección crítica y no compartan trabajo, un proceso debe esperar como máximo $(n-1)$ entradas diferentes en la sección crítica.

Para poder estimar la cantidad de referencias a memoria remota, se considera que cada proceso P_i accede a las variables *flag(i)*, *grupo(i)* y *oponente_compa(i, 1 ..n)* en forma local y el resto de los accesos en forma remota (NUMA). En la siguiente tabla se muestra una comparación entre los algoritmos con dos actores y con un actor, considerando en el *algoritmo con dos actores* la cantidad de accesos del actor proceso.

Casos	Alg. dos actores	Alg. un actor
P_i quiere acceder cuando hay un compañero y es el primero	4 accesos para ingresar 1 acceso para salir	9 accesos para ingresar 1 acceso para salir
P_i es el primer proceso y no tiene oponentes	no se puede determinar	3 + log(n) + (n-1) accesos para ingresar 3 accesos para salir

En las figuras 8 y 9, se muestran algunos gráficos relacionando la cantidad de procesos, la cantidad de accesos y la proporción que hay entre ellos para el algoritmo presentado de un actor.

En otros casos, no se puede estimar la cantidad de accesos, ya que se tienen esperas ocupadas sobre variables compartidas. Para poder estimar el peor caso, se debería adaptar el algoritmo para que todas las esperas ocupadas sean locales. La adaptación introduce mayor complejidad en el algoritmo e incluye nuevas variables.

6. Conclusión

En arquitecturas distribuidas, existen aplicaciones que conviven en el ambiente que no comparten recursos ni tareas en común, pero también existen aplicaciones que colaboran para resolver un problema, o se dividen en procesos para distribuir el trabajo entre los distintos nodos y obtener un mejor rendimiento (como por ejemplo, en cálculo numérico para resolver problemas basados en matrices).

El algoritmo presentado se basa en el modelo presentado en el caso (B) de la figura 1, utilizando memoria compartida. Está compuesto por un sólo actor *proceso*. Cada proceso selecciona el grupo de interés y comienza la competición para acceder al recurso. El protocolo permite que varios procesos trabajen concurrentemente con el recurso si seleccioan el mismo grupo de interés. Se compara el algoritmo presentado *con un actor* con el algoritmo *con dos actores*. El algoritmo con dos actores presenta un mejor performance en el caso óptimo que un proceso seleccione un grupo que está en la sección crítica y el primer proceso está en la misma, en los otros casos no se puede realizar una estimación ya que realiza espera ocupada sobre variables compartidas. En cambio en el algoritmo presentado en este artículo se puede obtener una cota para el primer proceso que ingresa en la sección crítica y no tiene oponentes, esta es *3 + log(n) + (n-1)* accesos a memoria y a mayor cantidad de procesos se tiende a necesitar n accesos a memoria.

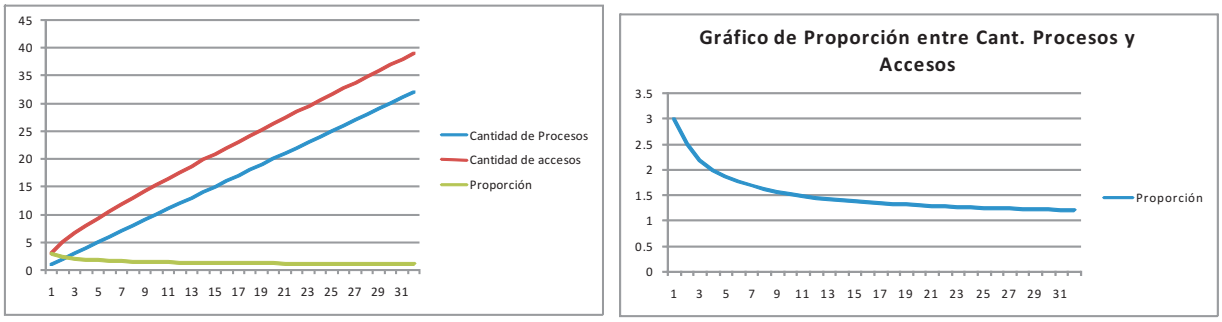

Figura 8: Con 32 procesos

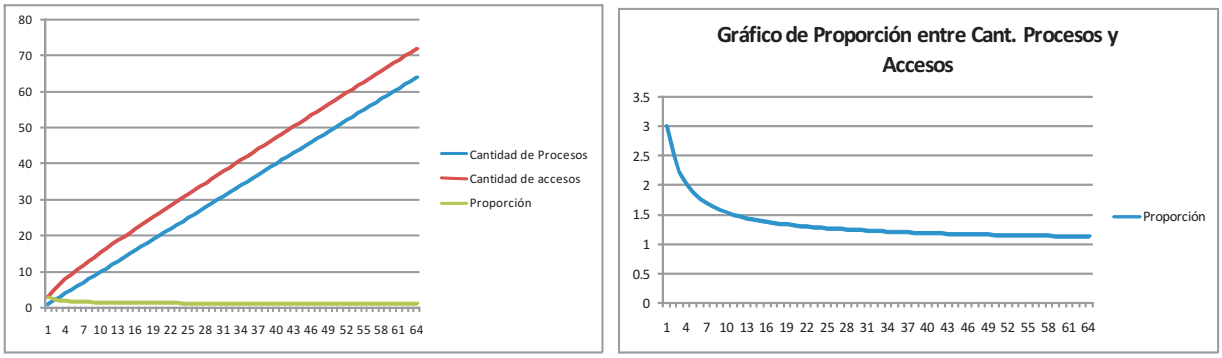

Figura 9: Con 64 procesos

Referencias

[1] M. Ben Ari. *Principles of Concurrent Programming*. Prentice Hall, Englewood Cliffs, 1982.

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

[3] 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.

[4] Karina Cenci, Jorge Ardenghi *Exclusión Mutua para Coordinación de Sistemas Distribuidos*. CACIC 2001.

[5] K. Cenci, J. Ardenghi *Algoritmo para Coordinar Exclusión Mutua y Concurrencia de Grupos de Procesos* CACIC 2002.

[6] Yuh-Jzer Joung *Asynchronous Group Mutual Exclusion (extended abstract)*. In Proc. 17 th. ACM PODC.

[7] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, Julio 1978.

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

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

[10] 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.

[11] Sape Mullender. *Distributed Systems*, 2da. Ed. 1993.

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

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

[14] Silberschatz, A., y Galvin, P. *Operating System Concepts*, 5ta. ed. Addison-Wesley, 1998.

[15] Jie Wu, *Distributed System Design*, 1999.