Algoritmo para Coordinar Exclusión Mutua y Concurrencia de 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

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 que permite que grupos utilicen el recurso, donde cada uno de los grupos está integrado por un conjunto de procesos y los grupos compiten por acceder al recurso. El protocolo está compuesto por grupos y por procesos, se basa en algoritmos distribuidos de memoria compartida asincrónica, a partir del algoritmo de Tournament de exclusión mutua para n procesos.

Palabras Claves: Sistemas Distribuidos - Exclusión Mutua - Concurrencia

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

1. Introducción

La exclusión mutua y la concurrencia son características fundamentales y esencialmente opuestas en sistemas distribuidos. En algunas aplicaciones tales como las que soportan trabajo cooperativo, es necesario imponer exclusión mutua sobre diferentes grupos de procesos, mientras que los procesos del mismo grupo comparten los recursos.

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.

Basándose en el modelo de memoria compartida, dónde los procesos se comunican mediante la escritura y lectura de variables compartidas, ¿cómo se maximiza la concurrencia? Intuitivamente, se puede pensar que se logra permitiendo que un proceso pueda acceder al recurso mientras el grupo esté activo y algún proceso interesado en el mismo trabajo se encuentre ocupando el recurso.

2. Planteo del Problema

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 compiten 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_2, P_3 y P_4 están actualmente vinculados al grupo G_2; 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_1 y P_6 están integrando el grupo G_3 que está compitiendo por alcanzar el permiso de utilizar el recurso.

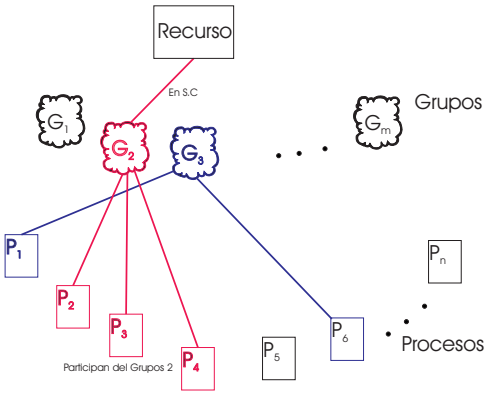

Figura 1

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.

- *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. Nociones del Algoritmo

El problema presenta dos tipos de actores: los *procesos* y los *grupos*, que se integran para competir por la utilización de un recurso. En el modelo, tenemos 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)*, 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 el *Esquema 1*, 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).

Proceso$_i$

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

Esquema 1

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.

Grupo$_k$

1. ..... {Región Resto}
2. Un proceso lo selecciona para trabajar en él. {Región de Entrada}
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. ..... {Región Resto}

Esquema 2

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

$\forall$ 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.

La variable *flag* indica si el grupo está compitiendo, y además en qué nivel se encuentra.

Grupo$_k$

Entrada$_k$
Si truncar(log(m)) = log(m) entonces

etapas = truncar(log(m))

sino

etapas = truncar(log(m)) + 1

fin si
para l= 1 hasta etapas hacer

flag(k) = l {representa a los diferentes niveles}

turn(comp(k,l)) = role(k,l)

Si role(k,l) $\neq$ 0 ó (k $\leq$ n-l) entonces

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)

```
Waitfor [∀ j ∈ oponentes(k,l) :
flag(j) < l] o [turn(comp(k,l))
≠ role(k,l)]

S.C. {en la sección crítica}

Salida_k

flag(k) = 0
```

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.

Para cada uno de los procesos:

```
Proceso_i
Entrada_i
Seleccionar el grupo (g_k)
Si inactivo(g_k) entonces

    lista[k,i] = <2, espera>

sino

    lista[k,i] = <1, espera>

fin Si

    Waitfor (flag(k) = etapas) y (∃ j:1..n,
    lista[k,j]=<2, en_cs>) (1)

lista[k,i] = <..,en_cs>
Sección Crítica
.....
Salida_i

lista[k,i] = <0, resto>
```

En el algoritmo se utiliza la variable *lista*, donde el primer índice especifica el grupo y el segundo índice el proceso. En la región de entrada, se selecciona el grupo en el cual va a participar. Luego verifica si el grupo está inactivo. Si la respuesta es afirmativa, es el primero que ingresa en el mismo, inicializa el nivel de la variable lista(k,i) en 2, de lo contrario lo hace en 1. Cuando se cumplen las condiciones de (1) significa que el grupo está en la región crítica y que este proceso puede acceder a ella.

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.

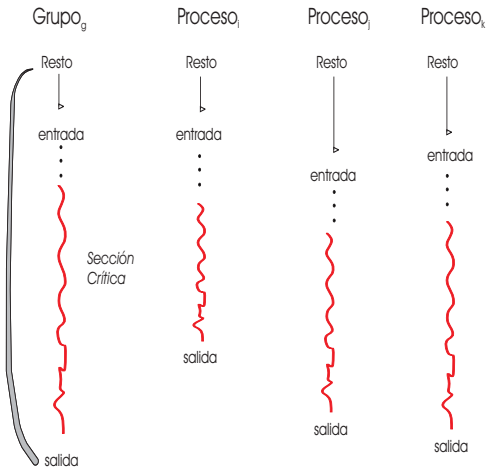

Figura 2

Como se observa en la figura 2, el grupo g se encuentra en la región crítica, el proceso i es el primer proceso del grupo, luego le sigue el proceso k que se agrega al grupo, entran a la región crítica en forma concurrente, cuando el proceso j selecciona el grupo g, ya está en la región crítica y como el primer proceso (proceso i) sigue trabajando en la misma entonces puede acceder también, y trabajar concurrentemente.

4. Algoritmo de Grupos para Exclusión Mutua

En el algoritmo presentado se utiliza el paradigma de memoria compartida distribuida sobre las variables de control utilizadas.

Variables Compartidas
∀ flag(i): 0 ≤ i ≤ (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 ≤ i ≤ (m-1), 0 ≤ j ≤ (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)

La variable *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. La variable *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 el esquema 3, se observa que cada grupo 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 región crítica. Indica a los procesos asociados al mismo que alcanzó el objetivo inicializando *flag(i)* a etapas + 1. Permanece en la región crítica mientras haya procesos activos en el grupo. Cuando todos los procesos salen de la región 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

Esquema 3

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 presentado en [2]); 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 el esquema 4, se muestra el ciclo que realizan cada uno de los procesos, verifican si el grupo está inactivo al momento de la selección, si es así se considera que es el primer proceso del 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>))
... *Sección Crítica*
Salida$_i$
lista(g,i) = <0, resto>

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

Esquema 4

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.

¿Qué sucedería si los procesos p_i y p_j seleccionan el mismo grupo g_k y este se encuentra inactivo?

Para p_i
flag(k) = 0 $\Longrightarrow$ el grupo está inactivo
lista(k,i) = <2, espera>

Para p_j
flag(k) = 0 $\Longrightarrow$ el grupo está inactivo
lista(k,j) = <2, espera>

Ocurre que dos procesos se consideran el primero del grupo. En el protocolo que controla el acceso del grupo al recurso, lo que se tiene en cuenta es que un proceso

lo seleccione. El paso de *Buscar_líder* es necesario, ya que puede haber procesos en espera por participar del grupo, que surgen mientras el mismo se encuentra en la región crítica, estos procesos no están habilitados para ingresar a la región crítica. El hecho de que haya 2 procesos líderes del grupo no afecta el protocolo de entrada del mismo, por lo tanto, se mantienen las condiciones de exclusión mutua y progreso. Lo que ocurre es que mientras alguno de estos 2 procesos se encuentre activo en el grupo, todos los procesos que quieren participar en el mismo pueden ingresar. Como se consideró que todos los procesos trabajan un tiempo finito, entonces el grupo no queda indefinidamente en la sección crítica, permitiendo que evolucionen otros grupos, evitando la inanición.

Si un proceso selecciona un grupo que tiene el acceso a la región 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 región crítica y vuelva a competir por el recurso, en el caso que todos los grupos estén compitiendo por acceder a la región, deberá esperar en el O(m+etapas) ciclos para acceder nuevamente a la misma, esto es esperar que entren los grupos restantes.

5. 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 modelo presentado, basado en memoria compartida, permite que procesos distribuidos que comparten recursos, como por ejemplo variables, datos coordinen el trabajo. Está compuesto por dos actores: *grupos* y *procesos*. Los procesos seleccionan el grupo en el cual van a trabajar cooperativamente y los grupos compiten por acceder al recurso. El protocolo permite que varios procesos trabajen concurrentemente con el recurso si pertenecen al mismo grupo, evitando que un grupo permanezca indefinidamente utilizando el recurso y garantizando la exclusión mutua y progreso.

Este algoritmo ha sido simulado en una red de estaciones de trabajo Sun y Digital Alpha.

Referencias

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

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

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

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

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

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

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

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

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