Aplicaciones Distribuidas: Coordinación y Sincronización

Karina M. Cenci * Jorge R. Ardenghi †

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

Introducción

Una aplicación distribuida está compuesta por un conjunto de procesos. Los procesos pueden cooperar para realizar una tarea o competir por la utilización de un recurso.

Un proceso cooperativo es aquel que puede afectar o ser afectado por otros procesos que se encuentran en ejecución en el sistema. Los procesos cooperativos pueden directamente compartir un espacio de dirección lógico (esto es, datos y código), o solamente comparte los datos a través de archivos.

El acceso concurrente a los datos compartidos puede resultar en un inconsistencia de los mismos. Se necesitan mecanismos para asegurar un ordenamiento en la ejecución de los procesos cooperativos que comparten un espacio de direccionamiento lógico, tal que la consistencia en los datos sea respetada.

Los mecanismos para asegurar el ordenamiento en la ejecución de procesos son: sincronización, exclusión mutua en las secciones críticas, alocación de recursos.

El proyecto en estudio está basado en el análisis, comparación y búsqueda de nuevas alternativas de algoritmos distribuidos que se pueden aplicar para soportar la exclusión mutua o cooperación entre los procesos.

Exclusión Mutua

La *exclusión mutua* maneja el problema de acceder a un único e indivisible recurso (como por ejemplo una impresora) que solamente puede soportar a un usuario a la vez, entre n usuarios U_1 .. U_n, o los conflictos resultantes de varios procesos compartiendo recursos.

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

Alternativamente se puede pensar éste como el problema de asegurar que ciertas secciones de código de programa sean ejecutadas en forma estrictamente exclusiva.

Un usuario con acceso al recurso es modelado estando en la región crítica, la cual es simplemente un subconjunto de sus estados posibles.

Cuando un usuario 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 usuario 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 usuario 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.

Condiciones de un Correcto Algoritmo

Para una dada colección de usuarios y para un sistema de memoria compartida A resolver el problema de exclusión mutua significa satisfacer las siguientes condiciones:

- Buena Formación: en cualquier ejecución, para cualquier i, la sub-secuencia que describe la interacción entre U_i y A está bien formada para i.
- Exclusión Mutua: no se alcanza un estado del sistema en el cual más de un usuario se encuentra en la región crítica C.
- Progreso: en cualquier punto de una ejecución imparcial:
    1. (progreso para la región de entrada): si al menos un usuario está en T y ningún usuario en C, en un punto posterior en el tiempo algún usuario entra a C.
    2. (progreso para la región de salida): si al menos un usuario está en E, en un punto posterior en el tiempo algún usuario entre a R.

La diferencia es que las condiciones de buena formación y exclusión mutua son propiedad de seguridad, mientras que la condición de progreso es una propiedad de vivacidad.

Un algoritmo regular/bueno de exclusión mutua debería satisfacer:

1. Libre de Interbloqueo: cuando la sección crítica (región crítica) está disponible, los procesos no deben esperar indefinidamente y alguno pueda entrar.
2. Libre de inanición: cada requerimiento a la sección crítica debería ser eventualmente garantizado.
3. Imparcialidad: los requerimientos serán otorgados basados en ciertas reglas de imparcialidad. Típicamente, está basado en los requerimientos de tiempo determinados por relojes lógicos. Inanición e imparcialidad están relacionados con lo que se denomina una condición fuerte.

En la realidad, inanición e imparcialidad pueden no ser críticas, en muchas situaciones en que se utiliza exclusión mutua, la contención entre los usuarios es poco frecuente y un usuario puede permitirse esperar hasta que todos los usuarios en conflicto obtengan un turno. La importancia de estas condiciones depende del grado de contención del recurso, como así también de la criticidad individual del programa usuario.

Tipos de Algoritmos de Exclusión Mutua según el Modelo de Memoria Compartida

Existen diferentes tipos de algoritmos de acuerdo a las propiedades que presentan, por ejemplo: algoritmos de exclusión mutua rápidos, algoritmos *adaptive* (adaptivos), algoritmos de exclusión mutua basados en el tiempo, algoritmos *nonatomic*.

En los algoritmos de exclusión rápidos, existe una gran diferencia en la complejidad de tiempo (complexity time) entre casos que están libres de contención y en los que se presenta contención. En los algoritmos de exclusión mutua *adaptive*, el incremento en la complejidad de tiempo en la contención es más gradual, está en función del número de procesos en contención. Los algoritmos de exclusión mutua basados en el tiempo sirven para mantener un grado de sincronización. No necesariamente todos los algoritmos de exclusión mutua que utilizan el modelo de memoria compartida pertenecen a las categorías mencionadas.

Las nociones de contención, para el caso de los algoritmos adaptivos, que se han considerado en la literatura son las siguientes:

- intervalo de contención (interval contention)
- punto de contención (point contention)

Estas nociones están definidas con respecto a la historia H. El *intervalo de contención* sobre H es el número de procesos que están activos en H, i.e, que se ejecuta fuera de su sección non-crítica en H (o sea están en una sección diferente a la de resto). El *punto de contención* sobre H es el número máximo de procesos que están activos en el mismo estado en contención.

Medidas en la Complejidad del Tiempo

Las consideraciones para las medidas en la complejidad de tiempo pueden ser varias:

- 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 remotas por un proceso en orden para ingresar y salir de su sección crítica.

Extensiones de Exclusión Mutua

El problema de la exclusión mutua tradicional se lo puede considerar como el caso 1-exclusión, en el cual un sólo proceso puede acceder a la sección crítica en un determinado instante de tiempo. El concepto de exclusión mutua se puede utilizar para sincronizar tareas en las cuales varios procesos pueden compartir un recurso, o un espacio. Por este motivo se comienza con el estudio de extensiones del concepto de exclusión mutua a κ-exclusión y exclusión mutua para grupos.

κ-exclusión

El problema de la κ-exclusión fue impulsado por Fischer, como una generalización al problema de exclusión mutua en el cual hasta κ procesos pueden estar al mismo tiempo en la sección crítica.

La propiedad de libre inanición es modificada para garantizar progreso en la fase incremento hasta k-1 ("face up to k-1")fallas indetectables de parada de procesos. Esto significa que hasta k-1 procesos fallan por parada (halting), cualquier proceso que no falla en su sección de entrada eventualmente ingresa en la sección crítica. En el caso que un proceso se para no inicializa ninguna de sus variables a los valores por defecto.

Exclusión Mutua para Grupos

Es otra generalización del problema de la exclusión que permite que múltiples procesos ejecuten su sección crítica simultáneamente. En este problema, los procesos se asocian para ejecutarse con una o más *sesiones*. Múltiples procesos pueden simultáneamente ejecutarse dentro de la misma sesión, pero diferentes sesiones no pueden estar activas en el mismo instante de tiempo.

Trabajo Futuro

Se está trabajando actualmente en el análisis y búsqueda de variaciones al problema de exclusión mutua tradicional para n procesos en un ambiente de memoria compartida distribuida, teniendo en cuenta para su implementación que las variables sean de multi-lectura y simple-escritura, ya que facilitan la misma. Algunos de los algoritmos que se han estudiado y analizado son Peterson para 2 procesos, Peterson para n procesos, Tournament, del Panadero.

Los resultados obtenidos sobre el problema inicial se han empezado a extender a la problemática de exclusión mutua para grupos, presentando un algoritmo para exclusión mutua para grupos sobre la base de una variación del algoritmo de Tournament. Se están preparando diferentes tipos de pruebas para realizar sobre distintas arquitecturas y sistemas operativos (PC, Sun, Alpha).

Referencias

[1] Yehuda Afek, Danny Dolev, Eli Gafni, Michael Merritt, Nir Shavit. A Bounded First-In, First-Enabled Solution to the l-Exclusion Problem. *ACM Transactions on Programming Language and Systems*, Mayo 1994.

[2] Yehuda Afek, Hagit Attiya, A. Fouren, G. Stupp, D. Touitou. Long-lived renaming made adaptive. *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing*, Mayo 1999.

[3] James H. Anderson, Yong-Jik Kim. Fast *and* scalable mutual exclusion. *Proceeginds of the 13th International Symposium on Distributed Computing*, Septiembre 1999.

[4] James H. Anderson, Yong-Jik Kim. Adaptive Mutual Exclusion with Local Spinning. *Proceedings of the 14th International Symposium on Distributed Computing*, Octubre 2000.

[5] James H. Anderson, Yong-Jik Kim. An improved lower bound for the time complexity of mutual exlcusion. *Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing*, Agosto 2001.

[6] Hagit Attiya, Vita Bortnikov. Adaptive and efficient mutual exclusion. *Proceedings of 19th Annual ACM Symposium on Principles of Distributed Computing*, Julio 2000.

[7] Karina Cenci, Jorge Ardenghi. Exclusión Mutua en la Implementación Memoria Compartida Asincrónica. *VI Congreso Internacional de Ingeniería Informática, ICIE 2000*, 26 al 28 de Abril 2000 - Facultad de Ingeniería - UBA.

[8] Karina Cenci, Jorge Ardenghi. Sobre Algoritmos Distribuidos de Exclusión Mutua para n procesos. *CACIC 2000*.

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

[10] Karina Cenci, Jorge Ardenghi. Algoritmo para Coordinar Exclusión Mutua y Concurrencia de Grupos de Procesos. *CACIC 2002*.

[11] Yuh-Jzer Joung. Asynchronous Group Mutual Exclusion (extended abstract). *Proceedings of the 17th. Annual ACM Symposium on the Principles of Distributed Computing*, 1998.

[12] Patrick Keane, Mark Moir. A Simple Local-Spin Group Mutual Exclusion Algorithm, *Technical Report* 1999.

[13] Yong-Jik Kim, James H. Anderson. A Time Complexity Bound for Adaptive Mutual Exclusion (Extended Abstract). *Proceedings of the 15th International Symposium on Distributed Computing* Octubre 2001.

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

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

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

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

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

[19] J.-H. Yang, James Anderson. A fast, scalable mutual exclusion algorithm. *Distributed Computing*, Agosto 1995.

[20] J.-H. Yang, James H. Anderson. A fast, scalable mutual exclusion algorithm. *Distributed Computing*, Agosto 1995.