SISTEMAS DISTRIBUIDOS

Práctica II

Problema de los lectores y escritores distribuidos

Viernes B Mañana Pareja 2

26/10/2023

ÁLVARO DE FRANCISCO NIEVAS, 838819@unizar.es

GUILLERMO BAJO LABORDA, 842748@unizar.es

Universidad Zaragoza

1542

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

1. Introducción

En esta práctica se ha implementado el algoritmo de Ricart-Agrawala generalizado para solucionar el problema del acceso a sección crítica en un sistema distribuido. En este caso el sistema implementa el problema de los lectores / escritores, en el cual todos los procesos tienen la misma responsabilidad.

Hemos modificado el algoritmo de Ricart-Agrawala para que use relojes vectoriales para establecer la relación "happens-before" entre un proceso que pide acceso a la sección crítica y el proceso que recibe la petición.

2. Implementación del algoritmo Ricart-Agrawala

El algoritmo de Ricart-Agrawala asume una red en la que N procesos distribuidos que no comparten memoria y se comunican exclusivamente mediante el **paso de mensajes**. La red de comunicación está **libre de errores**, pudiendo llegar estos fuera de orden.
Como todos las soluciones al acceso a sección crítica, el algoritmo cuenta con un pre-protocolo y un post-protocolo:

Pre-protocolo

El acceso se solicita mediante un mensaje a todos los procesos. Una vez que todos nos hayan concedido el acceso, podremos acceder.
Un proceso concederá acceso si:

- No quiere acceder a la SC
- Recibe una petición con menos timestamp que el suyo
- La matriz de excepciones nos dice que puede acceder sin poner en peligro el estado del sistema.

Esta última opción no forma parte del algoritmo, sino que la hemos implementado para nuestro problema.
Si se cumplen las condiciones, el proceso enviará inmediatamente un mensaje al proceso, concediendo acceso. En caso contrario guardaremos el proceso que nos ha pedido acceso.

Post-protocolo

Una vez hayamos salido de la SC, revisaremos si hemos postergado el acceso a algún proceso y le responderemos para que acceda a la SC.

En nuestra implementación del algoritmo, hemos utilizado dos elementos que no se incluyen en el algoritmo original, los relojes vectoriales y la matriz de exclusión.

Relojes vectoriales

Los relojes vectoriales son fundamentales para mantener un orden parcial de eventos en un sistema distribuido, lo que es esencial para garantizar la sincronización de procesos concurrentes. En este contexto, se han empleado relojes vectoriales para establecer una relación de orden total "happens-before" entre eventos en diferentes procesos.

La diferencia entre entre los relojes vectoriales y los relojes escalares radica en la capacidad de los relojes vectoriales para capturar el orden de eventos en múltiples procesos, lo que les permite establecer un orden parcial preciso y detectar relaciones "happens-before" en sistemas distribuidos. Los relojes escalares, en cambio, sólo pueden proporcionar información sobre eventos dentro de un proceso individual y carecen de la capacidad de establecer un orden total entre eventos en diferentes procesos.

Para la implementación, hemos utilizado la librería de relojes vectoriales “GoVector”, lo que permite registrar y comparar eventos en diferentes procesos en nuestro sistema. El reloj se actualiza en varias partes del código, como en la preparación de eventos y en la recepción de mensajes. Cuando un evento ocurre o se recibe un mensaje, se incrementa el componente correspondiente en el reloj vectorial local, lo que refleja el orden temporal de los eventos.

En esta parte es en donde más errores cometimos. En un principio no guardábamos el reloj que habíamos enviado a otros procesos en el pre-protocolo, lo que causaba incoherencias en la relación “happens-before”. Esto ocurría ya que cada vez que recibimos un mensaje se incrementa el reloj, por lo que el reloj que tenía nuestro proceso cambiaba constantemente. Para solucionar esto, cada vez que preparamos un mensaje de pre-protocolo guardamos el reloj que utilizamos para hacerlo, y así usarlo en la comparación.

Matriz de exclusión

En nuestro sistema de lectores/escritores tenemos dos operaciones, leer y escribir. La operación leer no cambia el estado del sistema, a diferencia de la operación escribir. Para minimizar el tiempo que los procesos pasan en SC, solo bloquearemos los procesos que puedan causar incoherencias en el estado. Estos son los escritores si hay un escritor en sección crítica. Para el resto de combinaciones de operaciones permitiremos el acceso a SC.
Construiremos una matriz de exclusión en la cual la única combinación de operaciones que se excluyen son “Read”, “Read”.

3. Descripción del sistema

Para N workers, lanzaremos N/2 escritores y N/2 lectores. No necesitaremos ningún proceso que arbitre la ejecución. El sistema distribuido de lectores/escritores está implementado utilizando tres módulos distintos: el algoritmo para la exclusión mútua (ra), un sistema de intercambio de mensajes (ms) y un gestor de ficheros para escribir y leer (gf).

Ya se ha detallado el funcionamiento del módulo para exclusión mútua, *ra*. A continuación se explicarán los otros dos componentes del sistema.

Gestor de ficheros

El gestor dispone de tres operaciones: *LeerFichero(), EscribirFichero(fragmento)* y *New(nom_fich)*. Una vez creamos una "instancia" de gf, leemos y escribimos en el fichero "nom_fich" utilizando las distintas funciones.

Sistema de intercambio de mensajes

Utilizamos el módulo proporcionado de intercambio de mensajes para comunicarnos con el resto de workers. Dado que la operación *Receive()* que nos proporciona es bloqueante, es decir, no seguimos con la ejecución hasta que recibimos un mensaje, lanzamos una gorutina que atiende el buzón. Esta rutina utiliza un canal por cada tipo de mensaje para redireccionarlo.
Para el proceso de envío de mensajes hemos utilizado la operación *Send()* del módulo en todo el programa, incluyendo el *ra.go*. Sabemos que no es una buena práctica ya que de esta forma atamos la implementación a una implementación concreta de sistema de intercambio de mensajes. No obstante, dada la reducida escala del problema y para aumentar la velocidad de desarrollo decidimos implementarlo de esta manera.

En nuestro sistema general intercambiaremos cuatro tipos de mensajes: Request, Reply, FileChange y Barrier. Los dos primeros tipos los utilizaremos en el algoritmo de Ricart-Agrawala para pedir y conceder acceso a la SC. Los dos últimos se utilizarán para intercambiar mensajes entre los procesos.
Los mensajes de tipo FileChange contienen los cambios realizados en el fichero por los procesos escritores, y se lo enviarán a todos los procesos del sistema para que actualicen sus respectivos ficheros.
El mensaje Barrier lo utilizaremos para comenzar la ejecución de forma sincronizada.