LECTORES Y ESCRITORES DISTRIBUIDO

SISTEMAS DISTRIBUIDOS - PRÁCTICA 2

Lizer Bernad Ferrando - 779035
Lucía Morales Rosa – 81690

---

1 INTRODUCCIÓN

En esta práctica se ha llevado a cabo la implementación distribuida del problema de los lectores y escritores haciendo uso del algoritmo de Ricart-Agrawala generalizado.

En esta aplicación, cada uno de los procesos lectores y escritores cuentan con su propio fichero de texto sobre el que podrán realizar sus funciones de lectura y escritura respectivamente.

Al ser un sistema distribuido puede darse el caso de que varios procesos quieran acceder a sus ficheros de forma simultánea. En el caso de los lectores, como el fichero no sufre modificaciones durante la lectura, se podrá permitir su acceso al fichero simultáneamente.

Sin embargo, como los escritores sí modifican su fichero al acceder a él, se deberá garantizar su acceso al mismo en exclusión mutua, impidiendo así que otro proceso interfiera y genere condiciones de carrera. Además, se deberán actualizar los ficheros de todos los procesos, tanto lectores como escritores, tras una escritura para garantizar que el contenido sea el mismo. Para ello, se hará uso de un gestor de ficheros.

Para garantizar esta exclusión mutua y la correcta sincronización entre procesos, se ha empleado el algoritmo de Ricart-Agrawala Generalizado cuyo diseño puede encontrarse en la sección siguiente.

2 DISEÑO DEL ALGORITMO

Para el diseño del algoritmo de Ricart-Agrawala se ha traducido el algoritmo original escrito en Algol, modificando algunos puntos para emplear relojes vectoriales en lugar de los relojes lógicos.

El algoritmo de Ricart-Agrawala cuenta con tres estados: preprotocolo, sección crítica y postprotocolo.

Para acceder a la sección crítica donde se encuentran las funciones que pueden generar condiciones de carrera, los procesos deberán haber realizado el preprotocolo donde se enviará una solicitud de acceso a la sección crítica al resto de procesos y se esperará la confirmación de acceso. Cuando esto ocurra, se ejecutarán las operaciones de la sección crítica y, al finalizar, se llevará a cabo el postprotocolo encargado de avisar al resto de procesos de la liberación de la sección crítica, de forma que otro proceso podrá acceder a ella.

En la Ilustración 1 se puede observar la máquina de estados resultante de las condiciones explicadas anteriormente.

Lizer Bernad Ferrando - **779035**
Lucía Morales Rosa - **816906**

Ilustración 1: Máquina de estados del algoritmo

A partir de esta máquina de estados se ha diseñado el diagrama de secuencia del sistema distribuido que se puede observar en la Ilustración 2.

Cabe destacar que, en este diagrama, se han empleado dos tipos de interacción entre procesos. En primer lugar, las interacciones ilustradas con una flecha negra representan llamadas a funciones, mientras que las interacciones representadas con una flecha roja simulan envíos de mensaje. Se ha decidido hacerlo así ya que esta forma permite abstraer el envío de mensajes y simplificar el diagrama para facilitar su comprensión.

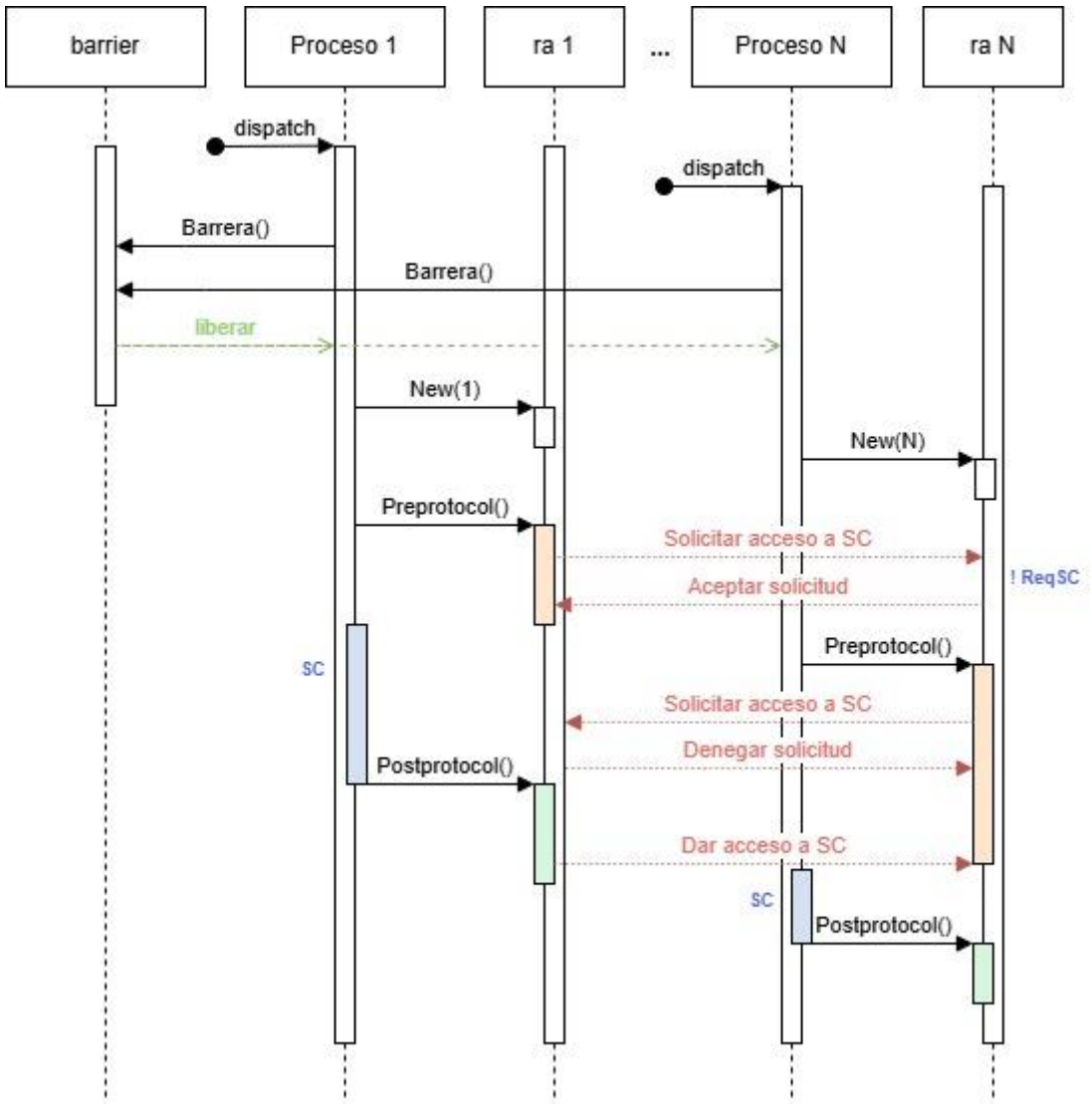

Ilustración 2: Diagrama de secuencia del problema de lectores-escritores distribuido con Ricart-Agrawala

Lizer Bernad Ferrando - **779035**
Lucía Morales Rosa - **816906**

Al inicio, tal y como se observa en el diagrama de secuencia, todos los procesos esperarán en una barrera para asegurar que estén en el estado inicial deseado antes de lanzar el algoritmo. Después, todos los procesos podrán pedir acceso a la sección crítica a través del preprotocolo y los demás procesos deberán aceptar o denegar la solicitud.

La decisión de aceptar o denegar una solicitud de acceso a la sección crítica vendrá dada por una serie de condiciones. Para negar la solicitud, el proceso que la recibe deberá querer tener acceso él mismo a la sección crítica y tener prioridad sobre el solicitante, es decir, que su reloj vectorial sea ancestro del reloj vectorial del proceso que solicita la entrada o que, en caso de empate, su identificador de proceso sea menor. Además, deberá comprobar que, al menos uno de los procesos, sea un proceso escritor. Para ello se hará uso de una matriz de exclusión que se explica con mayor detalle en la 3.2.

También se ha diseñado un diagrama de secuencia para representar la acción de escritura en la sección crítica de un proceso escritor hasta que comienza el postprotocolo. Tal y como se puede observar en la Ilustración 3, tras realizar la tarea de escritura, se enviará un mensaje de tipo "ActualizarFichero" con el contenido a añadir a todos los procesos para que modifiquen su propio fichero. Después, estos procesos responderán con un mensaje de tipo "AckActualizar", notificando al proceso escritor que ya se han actualizado. Cuando el proceso escritor reciba todos los *acks*, finalizará su sección crítica y comenzará su postprotocolo.

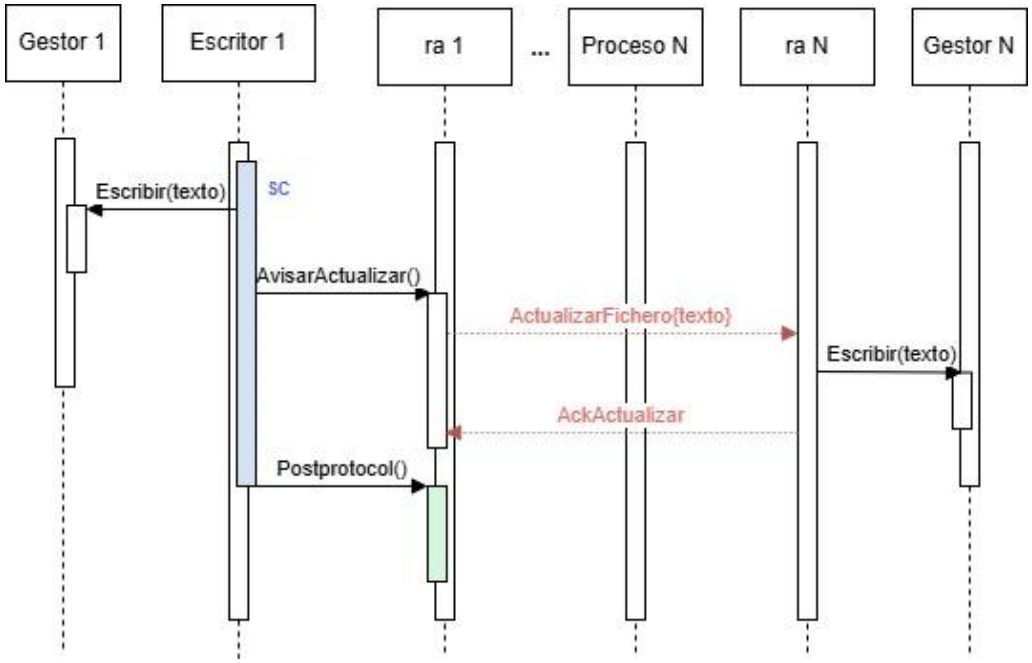

Ilustración 3: Diagrama de secuencia cuando un proceso escritor se encuentra en su sección crítica

3 ASPECTOS RELEVANTES DE LA IMPLEMENTACIÓN

Teniendo en cuenta el diseño de aplicación mostrado anteriormente, se han tomado algunas decisiones de implementación para poder llevarlo a cabo correctamente. Entre estos aspectos, destacan principalmente la estructura "*RASharedDB*", la matriz de exclusión que tendrá relevancia a la hora de decidir si se pospone un evento, un gestor de ficheros para llevar a cabo las distintas funciones de los lectores y escritores y la creación de unas *goroutines* para el manejo de los

mensajes recibidos de cada proceso. Además, también se ha creado un fichero, denominado "*Definitions.go*" con funciones auxiliares que emplearán otros módulos del sistema.

3.1 Estructura *RASharedDB* y relojes vectoriales

Para el correcto funcionamiento del algoritmo, se ha modificado la estructura "*RASharedDB*" proporcionada inicialmente.

Para empezar, se ha modificado el tipo de dato de los campos "*OurSecNum*" y "*HigSeqNum*" para que en lugar de representar relojes lógicos como lo hacían al ser enteros, pasen a actuar como relojes vectoriales al ser de tipo "*vclock.VClock*".

También se han declarado tres nuevos canales "peticion", "respuesta" y "ackActualizar" de tipo *Request*, *Reply* y AckActualizar respectivamente. Estos canales permitirán manejar los mensajes recibidos de forma sencilla y ordenada.

Por último, se han añadido otros tres campos. Dos campos son de tipo *string* y representan el identificador del fichero y el tipo de operación que realiza el proceso, pudiendo ser "Escribir" o "Leer". El otro dato es un puntero a un objeto de tipo "Fichero" definido en el módulo de gestor de ficheros del que se habla en mayor detalle en la 3.3.

3.2 Posponer evento y matriz de exclusión

Tal y como se ha comentado en secciones anteriores, cuando un proceso recibe una petición de acceso a la sección critica de otro proceso, puede concederle el acceso o posponerlo. Esta decisión se toma teniendo en cuenta diversos factores entre los que se encuentra el tipo de los procesos. Si ambos procesos son lectores, no habrá ningún problema si se produce un acceso simultáneo a la sección critica. Sin embargo, si alguno de los procesos es un proceso escritor, se deberán tener en cuenta los demás factores.

Para ello, se ha creado la siguiente matriz de exclusión:

$$matrizExclusion = \begin{pmatrix} false & true \\ true & true \end{pmatrix}$$

De esta forma se obtiene que:

- matrizExclusion[lector][lector] = false.
- matrizExclusion[lector][escritor] = true.
- matrizExclusion[escritor][lector] = true.
- matrizExclusion[escritor][escritor] = true.

Teniendo esto en cuenta, la decisión de aceptar o posponer una petición viene dada por la siguiente fórmula:

$$Posponer = reqSC \;\&\&\; tengoPrioridad \;\&\&\; matrizExclusion[tipoSolicitante][tipo]$$

Con esto, aseguramos que, para posponer una solicitud, el proceso deba querer tener acceso a la sección critica, prioridad de acceso sobre el proceso solicitante y cumplir con los requisitos de la matriz de exclusión.

Lizer Bernad Ferrando - **779035**
Lucía Morales Rosa - **816906**

3.3 Gestor de ficheros

El gestor de ficheros es el responsable de interactuar con los ficheros de los procesos. Este gestor tendrá las operaciones de creación, lectura y escritura donde se realizará la tarea especificada, comprobando anteriormente la existencia del fichero en el caso de la lectura y la escritura. Deberá tenerse en cuenta que, en caso de ser una escritura, esta se realizará al final del fichero y no se sobre escribirá.

Además, contará con un objeto de tipo "Fichero" que contendrá el nombre del fichero.

3.4 *Goroutines* de manejo de mensajes

Para el manejo de recepción de mensajes se han empleado *goroutines* de forma que los distintos tipos de mensajes se gestionen de forma simultánea.

Para ello, se ha creado una función "recibirMensaje" que atenderá de forma infinita a la recepción de mensajes. Cuando reciba alguno, lo clasificará por tipo, empleando los canales definidos en la estructura "*RASharedDB*" definida en la 3.1. Estos mensajes podrán ser de tipo *Request*, *Reply*, ActualizarFichero y AckActualizar.

Cuando el mensaje recibido sea de tipo *Request* se enviará por el canal "peticion" y se tratará en la función de manejo de peticiones, donde se actualizará el reloj y se pospondrá o aceptará la petición.

Si el mensaje es de tipo *Reply*, se enviará al canal "respuesta" donde se decrementará el numero de respuestas pendientes y se notificará que se han recibido todas las respuestas si es el caso.

Por otro lado, si el mensaje es de tipo ActualizarFichero, se realiza una llamada a la función de escritura del gestor de ficheros para añadir el texto recibido en el mensaje al fichero del proceso y, a continuación, se envía un mensaje de tipo AckActualizar.

Finalmente, cunado se reciben mensajes de tipo AckActualizar, se envía al canal homónimo y permitirá actualizar el número de respuestas esperadas de *ack* del proceso para que cuando llegue la última, el escritor pueda salir de la sección crítica.

3.5 Módulo com

Por último, se ha creado un módulo "com" a partir del proporcionado en la práctica anterior. Este módulo cuenta con un fichero "*definitions.go*" que incluye la definición de la función "*CheckError*" para manejar los errores críticos, una función para crear números aleatorios y una de depuración. Esta función de depuración creará un fichero "Depuracion.txt" si no existía anteriormente o lo sobre escribirá si es una nueva ejecución del sistema para almacenar las salidas de texto de depuración de cada nodo.

Por otro lado, tendremos un fichero "*barrier.go*" con la implementación de la barrera distribuida mencionada anteriormente que permitirá asegurar que los procesos se encuentran operativos para dar comienzo al algoritmo distribuido.