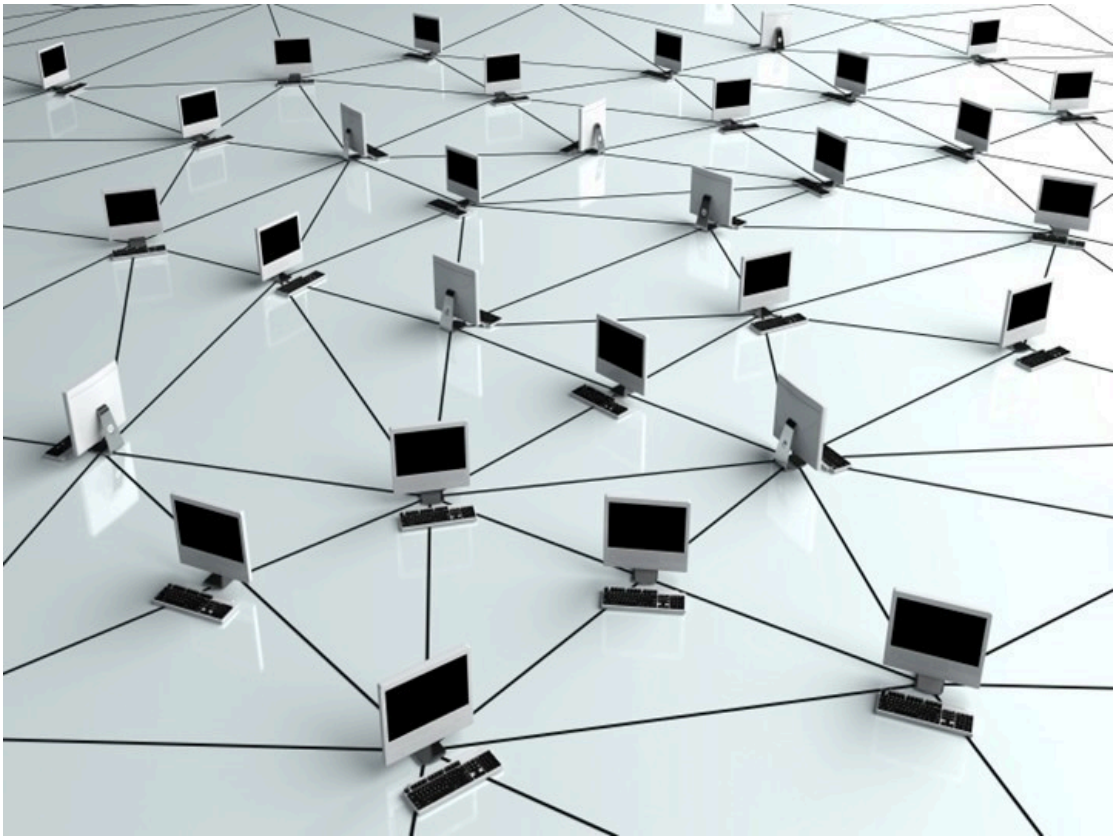
MEMORIA
SISTEMAS DISTRIBUIDOS

PRÁCTICA DE LABORATORIO 2

GRUPO TARDES 3-3



CURSO 2024 - 2025

Adrián Nasarre Sánchez 869561

Héctor Lacueva Sacristán 869637

ÍNDICE

1. Introducción

En esta práctica se implementa una solución al problema de los lectores y escritores en un entorno distribuido, utilizando el **algoritmo de Ricart-Agrawala generalizado** con relojes vectoriales. El objetivo principal es garantizar la correcta concurrencia en el acceso a ficheros de texto por parte de múltiples procesos distribuidos, tanto lectores como escritores.

2. Descripción del problema

2.1. Lectores y escritores en sistemas distribuidos

El problema de lectores y escritores plantea que múltiples procesos lectores pueden leer un archivo simultáneamente, pero solo un proceso escritor puede escribir en él. En un entorno distribuido, es necesario que todos los procesos compartan el mismo estado del archivo tras cada operación de escritura, de modo que las copias de los ficheros sean consistentes entre todos los nodos.

2.2. Algoritmo de Ricart-Agrawala

El **algoritmo de Ricart-Agrawala** original se diseñó para resolver el problema de exclusión mutua en un entorno distribuido. Para esta práctica, se ha extendido dicho algoritmo para gestionar múltiples tipos de operaciones (lectura y escritura), empleando relojes vectoriales para garantizar la sincronización entre los procesos.

3. Diseño de la solución

3.1. Arquitectura del sistema

El sistema está compuesto por:

- **N procesos lectores**: Acceden al contenido de los ficheros de manera concurrente.
- **M procesos escritores**: Actualizan el contenido de los ficheros, asegurando que las modificaciones se propaguen correctamente a todos los procesos.

Cada proceso cuenta con una copia local del fichero de texto y se coordina con los demás procesos mediante el algoritmo distribuido.

3.2. Algoritmo de Ricart-Agrawala con relojes vectoriales

El algoritmo ha sido modificado para funcionar con **relojes vectoriales** en lugar de relojes lógicos simples. Esto permite que los procesos comparen los estados de manera más

precisa en un entorno donde existen múltiples tipos de operaciones. Para ello, se modificó el fichero ra.go de la siguiente manera:

- Se modificó la clase RAShareddDB, que inicialmente estaba diseñada para relojes lógicos, eliminando los atributos OurSeqNum y HighSeqNum y añadiendo un logger de tipo *govec.GoLog que contiene la información del reloj vectorial local.
- También se implementó una función que recrea la relación total happens-before sobre dos relojes vectoriales para poder gestionar las peticiones de otros procesos correctamente.

3.3. Algoritmo de Ricart-Agrawala con relojes vectoriales generalizado para el problema de los lectores y escritores

A continuación, el algoritmo ha sido modificado para resolver el problema de los lectores y escritores. Para ello, se modificó el fichero ra.go de la siguiente manera:

- Se modificó la clase RASharedDB, añadiendo un atributo que es capaz de mapear los roles de dos procesos dando como resultado false si pueden ejecutarse en paralelo (cuando dos procesos son lectores) o true en caso contrario.
- A su vez, se añadió un atributo que representa qué tipo de tarea realiza el proceso, bien sea “write” (si es escritor) o “read” (si es el lector).

Asimismo, se han añadido otros dos atributos, ambos canales que distribuyen los mensajes recibidos de tipo Request y de tipo Reply a las funciones que los manejan.

4. Implementación en Go

4.1. Paquetes

4.1.1. Paquete mf

Este paquete proporciona funciones para el manejo de archivos, permitiendo **crear**, **leer** y **escribir** archivos de texto. Las funciones gestionan los errores de manera sencilla y, en caso de que ocurra un fallo, finalizan la ejecución del programa.

4.1.2. Paquete ms

El paquete ms implementa un **sistema de mensajería asíncrono** inspirado en el **Modelo Actor**. Los procesos se comunican mediante el envío y la recepción de mensajes utilizando **TCP**, lo que permite la interacción entre los procesos distribuidos de manera directa y sin bloqueo.

Cada proceso tiene un identificador único y envía mensajes a los demás procesos mediante su dirección IP y puerto, que se leen desde un archivo de configuración.

4.1.3. Paquete ra

Este paquete contiene la implementación del **algoritmo de Ricart-Agrawala Generalizado**. El algoritmo está diseñado para permitir que los procesos coordinen sus accesos a los archivos distribuidos, asegurando que no ocurran **condiciones de carrera** ni **inconsistencias**.

El paquete ra se basa en:

- **Solicitudes y respuestas**: Utilizadas para coordinar la entrada y salida de las secciones críticas.
- **Relojes vectoriales**: Se utilizan para asegurar que las operaciones en el sistema distribuido ocurran en el orden correcto, incluso cuando los procesos no comparten un reloj físico.
- **Canales de Go**: Utilizados para la comunicación entre goroutines y para recibir permisos o solicitudes.

4.1.4. Paquetes escritor y lector

Los paquetes escritor y lector implementan los **procesos escritores** y **lectores** que interactúan con el sistema de archivos distribuido. Ambos procesos utilizan el protocolo de Ricart-Agrawala para coordinar el acceso a sus archivos locales y compartir los cambios con los demás procesos.

4.2. Secuencia de Operaciones

4.2.1. Lectura (lector.go)

1. El proceso lector se inicializa y crea su propio archivo local.
2. Envía mensajes a los demás procesos anunciando una barrera de sincronización.
3. Antes de leer el archivo, solicita acceso a la sección crítica enviando una **Request** a los otros procesos.
4. Cuando recibe todas las respuestas necesarias, procede a leer el archivo.
5. Una vez leída la información, envía las respuestas diferidas (si las hubiera) y sale de la sección crítica.

4.2.2. Escritura (escritor.go)

1. El proceso escritor se inicializa y crea su propio archivo local.
2. Envía mensajes a los demás procesos anunciando una barrera de sincronización.
3. Solicita acceso a la sección crítica antes de escribir, enviando una **Request** a los otros procesos.
4. Cuando recibe todas las respuestas, actualiza el archivo local y distribuye la actualización a los demás procesos.
5. Finalmente, envía las respuestas diferidas y sale de la sección crítica.

4.3. Manejo de la Concurrencia

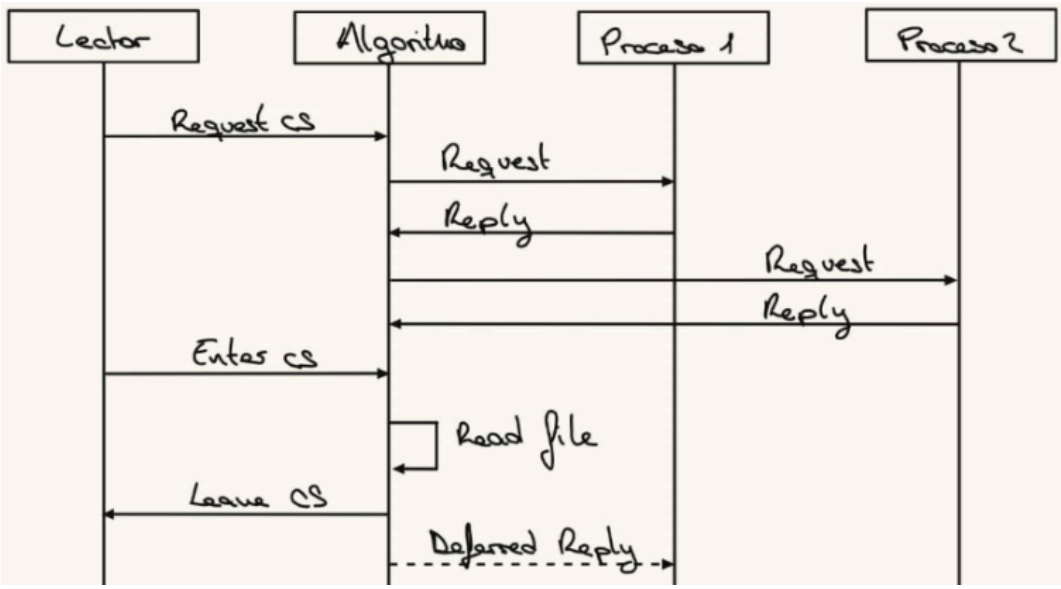
La concurrencia se maneja mediante:

- **Canales Go** para la recepción de mensajes y la espera de permisos.

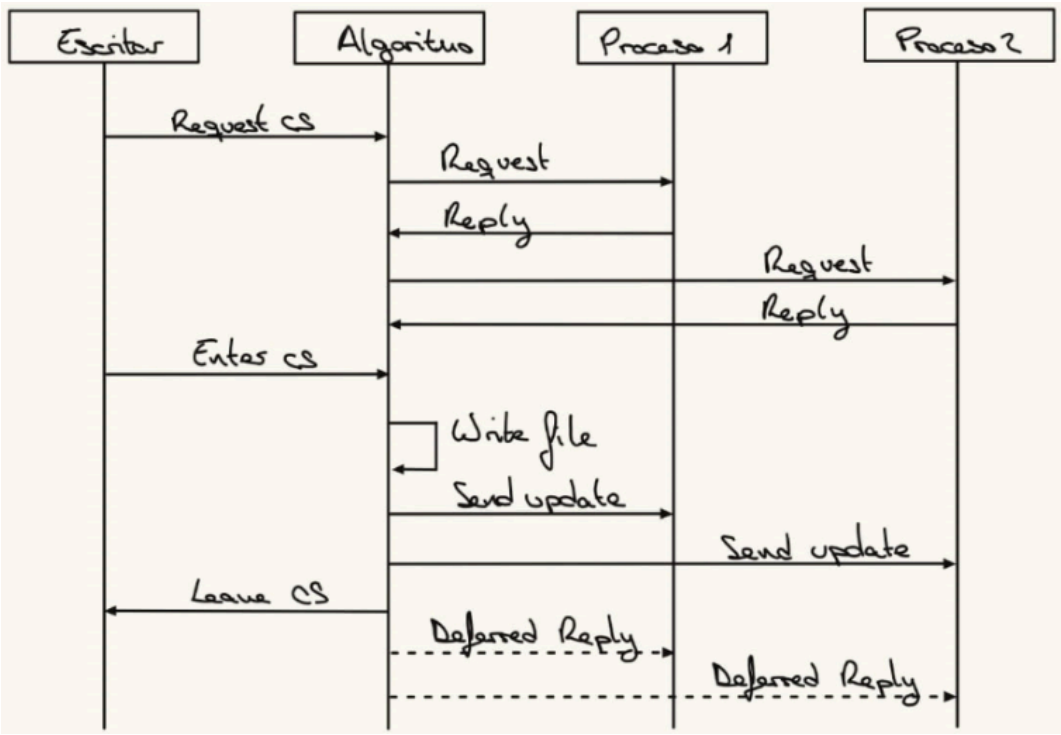
- **Mutexes** para proteger el acceso a las variables compartidas y evitar condiciones de carrera.
- **Relojes vectoriales** para asegurar el correcto orden de las operaciones distribuidas.

5. Diagrama de Secuencia

5.1. Diagrama de Secuencia para el Proceso Lector



5.2. Diagrama de Secuencia para el Proceso Escritor



6. Conclusiones

La implementación del algoritmo de **Ricart-Agrawala Generalizado** en Go ha permitido gestionar de manera eficiente y correcta la concurrencia entre múltiples procesos distribuidos. Las pruebas realizadas muestran que los procesos pueden leer y escribir archivos de manera concurrente, garantizando que las actualizaciones se propaguen correctamente entre los nodos del sistema.

7. Referencias

1. Ricart, G., & Agrawala, A. K. (1981). *An optimal algorithm for mutual exclusion in computer networks*. Communications of the ACM, 24(1), 9-17.
2. GoVector: https://github.com/DistributedClocks/GoVector
3. ShiViz: https://bestchai.bitbucket.io/shiviz