Entrega Práctica 2 - Sistemas distribuidos

Problema de los Lectores y Escritores Distribuido

Autores:

Fernando Pastor Peralta (897113)

Javier Murillo Jimenez (897154)

Fecha de entrega: 16/10/2025

Grupo: Pareja:1-6

Índice

Descripción del problema:

El problema a resolver en esta práctica consiste en implementar un sistema distribuido de lectores y escritores donde múltiples procesos acceden concurrentemente a un recurso compartido (en este caso un fichero de texto, guardado en cada dispositivo de forma local), garantizando lo siguiente:

- Múltiples lectores pueden acceder al fichero simultáneamente.
- Los escritores requieren acceso en exclusión mutua.
- No existe un coordinador por lo que todos los procesos tienen la misma responsabilidad en el sistema.
- Todas las copias locales deben converger al mismo estado después de cada escritura.

Para asegurarnos de que todo eso se cumple es necesario que se cumplan las siguiente propiedades:

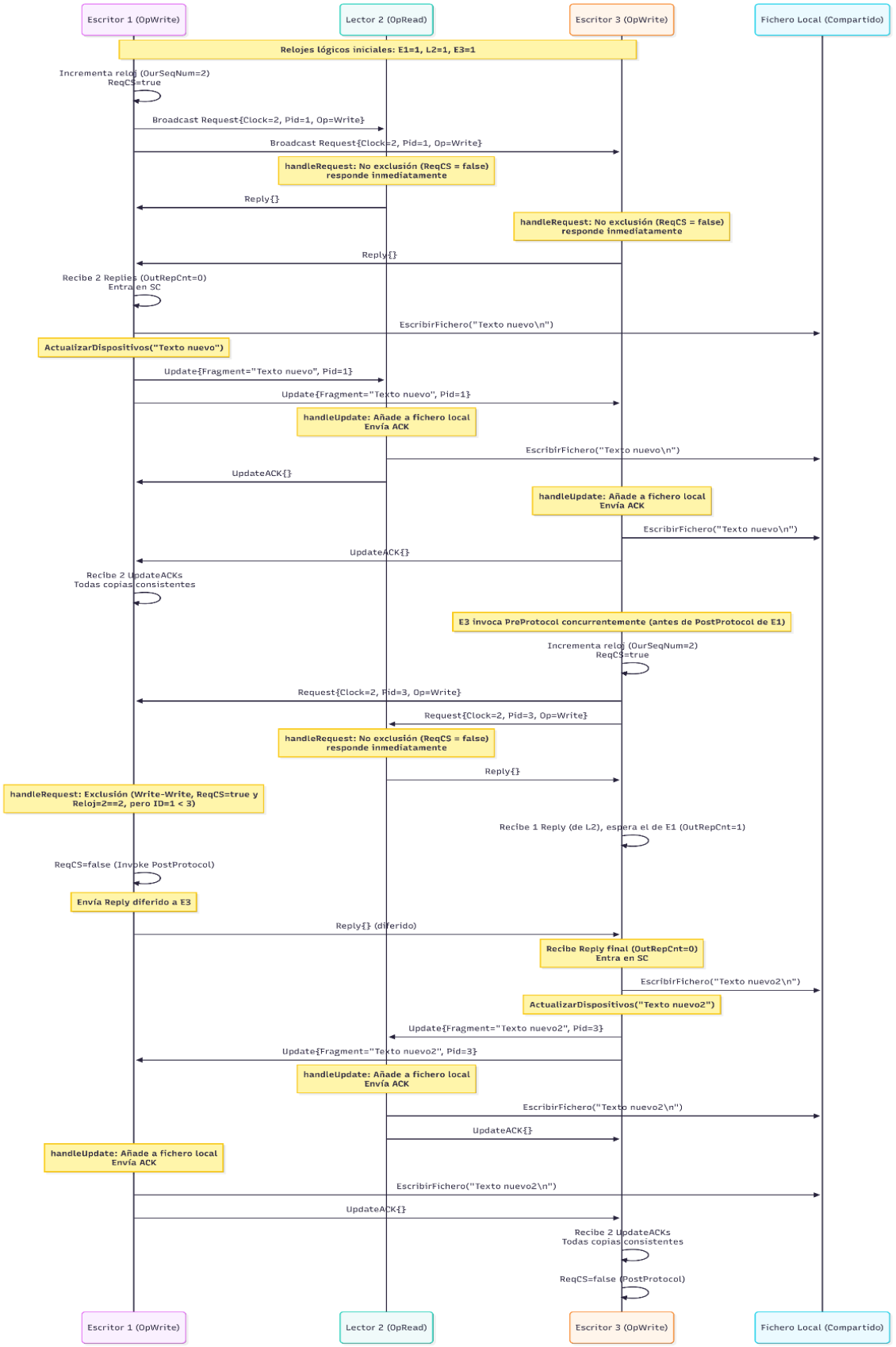
- Exclusión mutua entre escritores: Dos escritores no pueden acceder simultáneamente a la sección crítica.
- Exclusión mutua entre lector y escritor: Un lector y un escritor no pueden acceder simultáneamente a la sección crítica.
- Concurrencia entre lectores: Múltiples lectores pueden leer concurrentemente el fichero.

También aseguramos las siguientes propiedades de vivacidad:

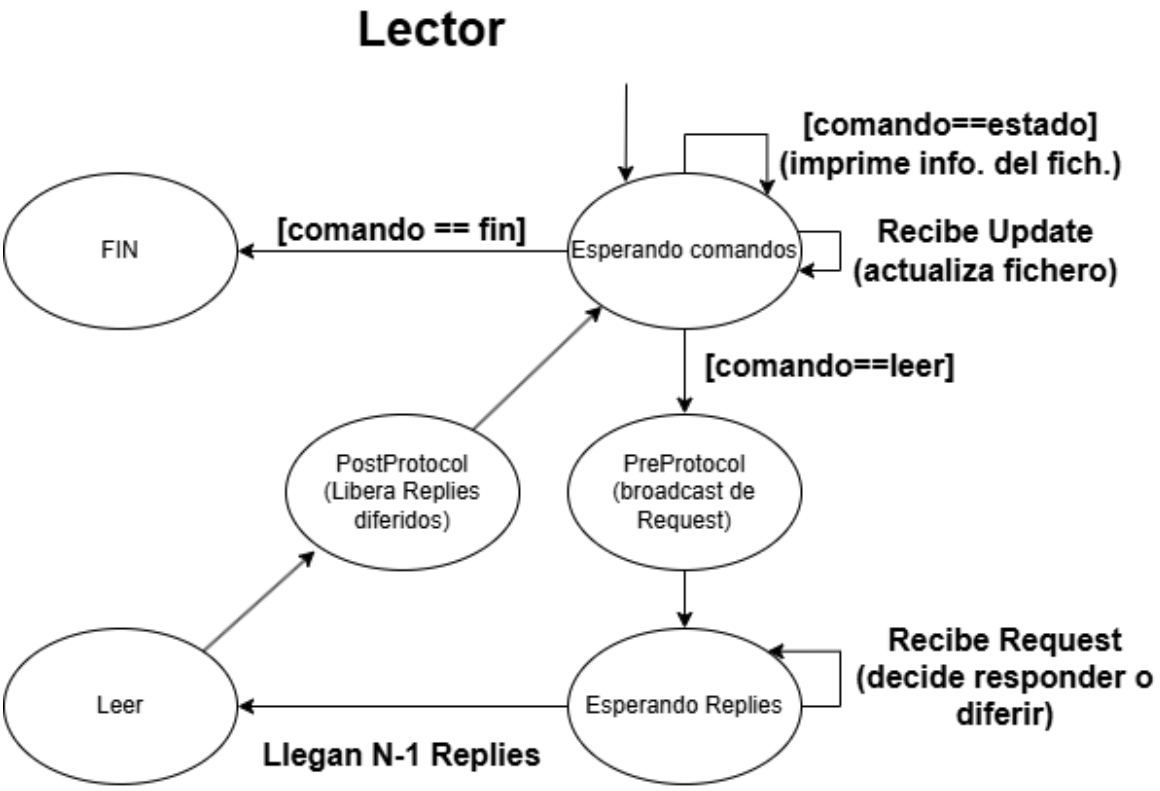
- Ausencia de inanición: Tanto lectores como escritores eventualmente obtendrán acceso a la sección crítica.
- Ausencia de interbloqueo: El sistema nunca entra en un deadlock.

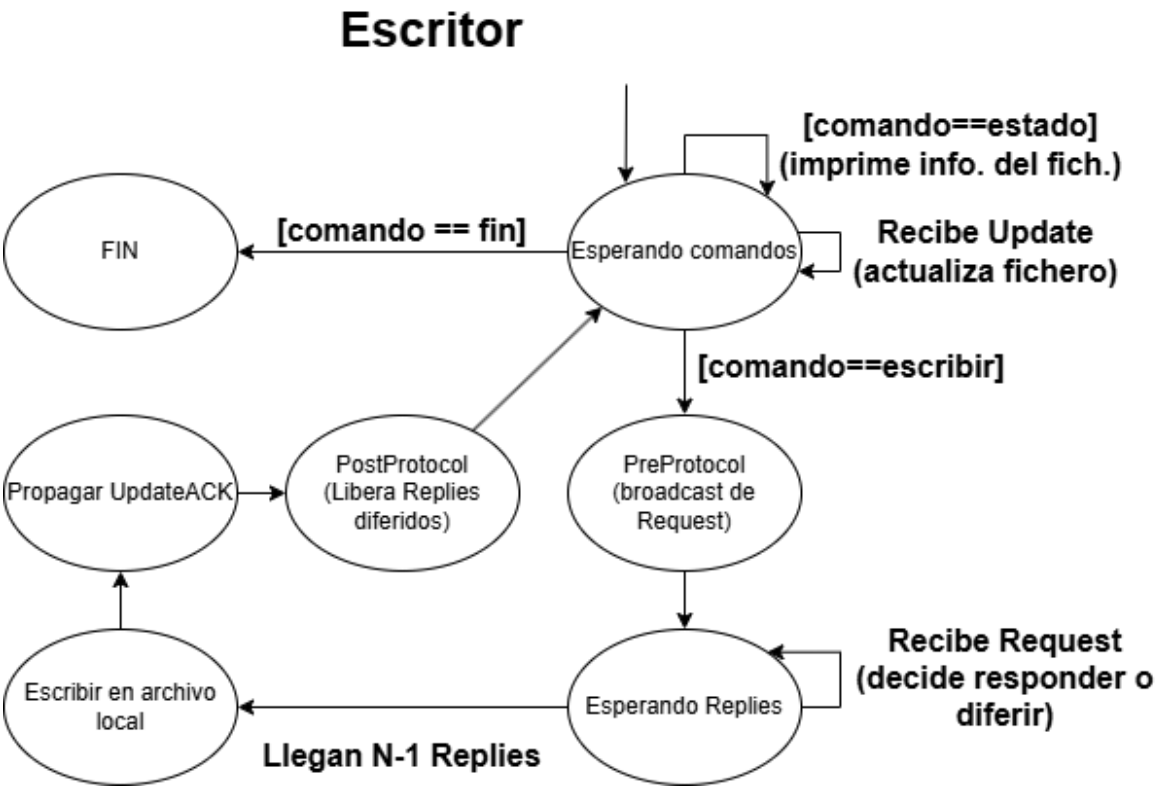
Diseño del algoritmo:

Diagrama de secuencias:



Máquina de estados:





Implementación:

Para la implementación del algoritmo, hemos extendido el algoritmo original de Ricart-Agrawala para manejar múltiples tipos de operaciones, para ello, utilizamos:

▸ Una base de datos común a todos los procesos distribuidos, basada en relojes lógicos, información del proceso local, booleano para indicar peticiones de entrada a la sección crítica, mutex y canales para recepción de mensajes:

```Go
type RASharedDB struct {
	OurSeqNum  int              // Número de secuencia del proceso
	HighSeqNum int              // Número de secuencia más alto visto
	OutRepCnt  int               // Número de replies que faltan para entrar a la SC
	ReqCS      bool             // Indica si el proceso está pidiendo acceso a la SC
	RepDefd    []bool           // Para cada proceso, si se le debe un reply
	MS         ms.MessageSystem // Sistema de mensajes
	done       chan bool        // Para terminar los procesos
	chrep      chan bool        // Canal para recibir los replies
	Mutex       sync.Mutex       // Mutex para proteger concurrencia sobre las variables

	id             int        // Id del proceso
	MyOp           int        // Operación actual (leer o escribir)
	N              int        // Número de procesos
	exclude         [2][2]bool // Matriz de exclusión mutua entre operaciones de lectura y escritura
	chUpdateACK     chan bool  // Canal para recibir ACKs de los dispositivos para confirmar que han actualizado su fichero
	archivo_EscLec string     // Fichero en el que escribir o leer
}
```

▸ Una matriz de exclusión, que forma parte de la base de datos antes mencionada, esta define las reglas de concurrencia entre lectores y escritores:

```Go
exclude: [2][2]bool{
	/*         read   write*/
	/*read*/  {false, true},
	/*write*/ {true,  true},
}
```

En esa matriz podemos ver que entre lectores no se excluyen pero en el resto de casos sí (lector-escritor, escritor-escritor).

▸ Receptor de mensajes mediante goroutine “receiver”, creada al momento de crear la base datos locales. Se ocupa de recibir mensajes, identificarlos y procesar la información dependiendo del tipo de mensaje.

▸ Paso de mensajes:

```go
type Request struct {
	Clock int //reloj lógico del proceso que envia el mensaje
	Pid   int //id del proceso que envia el mensaje
	Op    int //operación (leer o escribir)
}

type Reply struct{}

type Update struct {
	Fragment string
	Pid     int //id del proceso que envia el mensaje
}

type UpdateACK struct{}
```

Tipos de mensajes implementados:

- Request para la petición de acceso a la sección crítica (Clock, Pid, Op).
- Reply para la autorización de entrar a la sección crítica.
- Update para la propagación de cambios en el fichero
- UpdateACK para la confirmación de la actualización del fichero (Fragment, Pid).

Para pedir la sección crítica creamos el PreProtocol, que pone ReqCs a true, solicitando la sección crítica, actualiza el reloj, manda broadcast de Request para indicar que queremos entrar la SC y bloquea el proceso hasta recibir Reply de todos los dispositivos.

Los Reply se mandan automáticamente si no estamos solicitando la sección crítica, o no nuestra operación no tiene exclusión mutua con la operación que se está solicitando hacer, o si nuestro reloj indica mayor tiempo (o igual pero con ID menor) que el reloj del proceso que envió el mensaje. En caso contrario, se diferirá el Reply, y solo se mandará una vez que se ejecute nuestro PostProtocol, que indica que ya hemos salido de nuestra sección crítica (ReqCS = false).

Por otro lado, al tener que actualizar todos los ficheros locales cada vez que se escribe, tenemos que asegurarnos de que la actualización de los ficheros se ha realizado en todos los dispositivos antes de que cualquier otro dispositivo empiece a escribir, ya que si no, podríamos tener texto faltante o desordenado en el fichero de alguno de los dispositivos distribuidos.

Para asegurarnos de esto, además de los mensajes de Request para pedir acceso a la sección crítica y Reply para concederla, creamos dos tipos de mensajes más, para asegurar la actualización de ficheros locales en todos los dispositivos antes de salir de la exclusión mutua distribuida, estos son: Update, que se manda a todos los dispositivos distribuidos para decirles que actualicen con un determinado texto su fichero local, y otro mensaje UpdateACK, que es mandado al dispositivo que escribió en primer lugar, dispositivo que está a la espera de que le lleguen N-1 UpdateACK, ya que cuando llega el último mensaje de este tipo, nos hemos asegurado de que ya están todos los ficheros locales actualizados correctamente.

De todo este paso de mensajes para actualizar ficheros locales, se encarga la función ActualizarDispositivos, que lanza los mensajes de Update a todos los dispositivos y espera a que le lleguen N-1 UpdateACK. Se lanza cada vez que se hace la operación de escritura, entre el PreProtocol y el Postprotocol.

▸ Después se ha diseñado los algoritmos propios de los escritores y lectores, donde se crean las bases de datos, se pueden habilitar logs, se muestra una serie de comandos que se pueden ejecutar y se llaman a PreProtocol y PostProtocol al leer o escribrir (y a ActualizarDispositivos al escribir).

Pruebas:

Para las pruebas del algoritmo se ha creado un sistema de logs que permite ver con detalle el estado del sistema en cada momento, puede ser activado a la hora de llamar a la ejecución del lector o escritor poniendo “log” como último argumento de la llamada a ejecución. Este sistema, junto al comando que se puede poner en cualquier momento para ver el estado del documento compartido (“estado”), permite saber exactamente qué ha pasado y si la ejecución de las secuencias de acciones ha sido correcta o no.

Con este método para ver el estado del sistema, hemos hecho una batería de pruebas, en las que hemos puesto un temporizador de 5 segundos comentado después de la llamada a PreProtocol, tanto en el lector, como en el escritor. De esta forma, comentando y descomentando este temporizador, podremos ver si las llamadas que hagamos desde otros lectores o escritores sufren de exclusión mutua como deberían, o si hay algún comportamiento inesperado.
Las pruebas que hemos hecho incluyen:

- Comprobación de la existencia de exclusión mutua cuando un escritor intenta escribir mientras otro escritor escribe.
- Comprobación de la existencia de exclusión mutua cuando un lector intenta leer mientras un escritor escribe.
- Comprobación de la existencia de exclusión mutua cuando un escritor intenta escribir mientras un lector lee.
- Comprobación de la no existencia de exclusión mutua cuando un lector intenta leer mientras otro lector lee también.

Tras comprobar el correcto funcionamiento de las pruebas, podemos concluir que el sistema funciona bien.