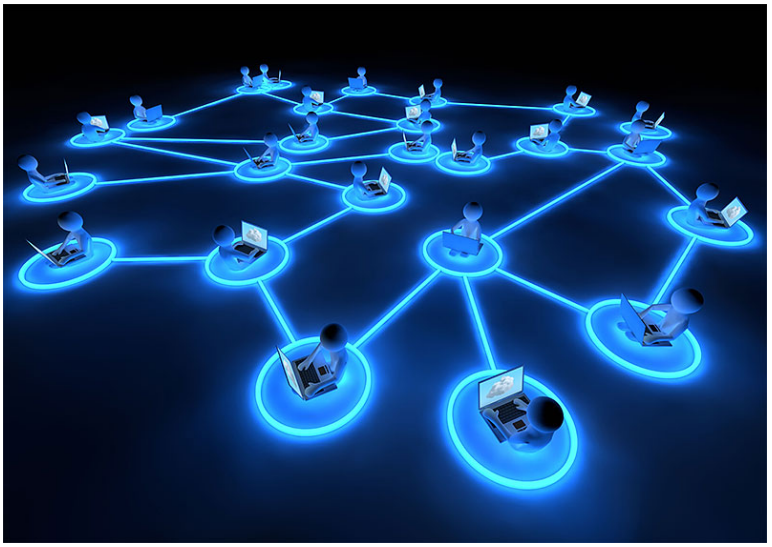
Práctica 2: Problema de los Lectores y Escritores Distribuido



Sistemas Distribuidos
Universidad de Zaragoza, curso 2022/2023

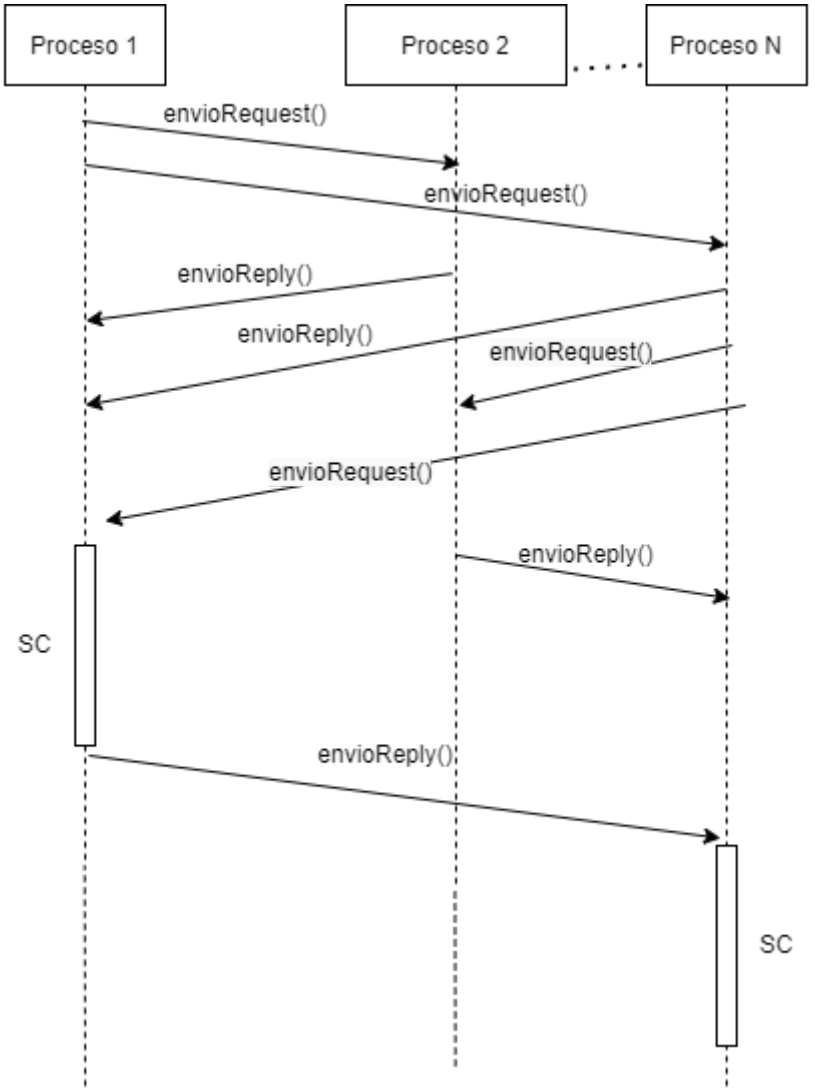
Ayelen Nuño Gracia 799301
Loreto Matinero Angusto 796598

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Descripción del problema

En esta práctica se va a realizar una implementación del algoritmo de Ricart-Agrawala mediante la utilización de relojes vectoriales, implementados a partir de la utilización de la librería GoVector, la cual ayuda tanto a comprobar el correcto funcionamiento, como a preparar el envío y recibo del reloj vectorial.

Algoritmo Ricart-Agrawala



La implementación de este algoritmo se ha llevado a cabo siguiendo la especificación del algoritmo en lenguaje Algol proporcionado para la realización de la práctica.
En primer lugar se han añadido las siguientes variables al código proporcionado:

- **Me:** número entero donde cada proceso guarda su pid
- **N:** número entero total de procesos que existen en la red
- **op_type:** entero que recibe los valores 0 (si el proceso es de tipo lector) ó 1 (si el proceso es de tipo escritor)

- **logger *govec.GoLog**: variable necesaria para la utilización de las funciones de la librería GoVector.

Además se ha añadido un nuevo tipo de dato **RelojVector**, el cual permite recoger la información proporcionada por las funciones de la librería GoVec, incluyendo los vectores de relojes y sus modificaciones.

Para la implementación del algoritmo se ha dividido el código en dos funciones, la primera de ellas llamada **PreProtocol**, se encarga de gestionar el acceso a la sección crítica, para ello envía a los N-1 procesos un mensaje del tipo request, en la que envía, su reloj, su id y el tipo de proceso que es (ya sea lector, o escritor).
En esta misma función nos quedaremos esperando a que cada uno de los N-1 procesos nos envíen su permiso, a través del canal chrep.

La segunda función es el **PostProtocol**, esta función la ejecuta un proceso justo cuando sale de su sección crítica, en ella el proceso que sale de la sección crítica mandará un Reply{} a todos los demás procesos indicando así que ya tienen su permiso para acceder a la sección crítica.

En medio de estas dos secciones encontramos el acceso a la sección crítica, donde los procesos escritores y/o lectores realizan su función.
El problema de los lectores, y de los escritores es el siguiente: cada proceso, independientemente de su tipo, va a tener una copia de un fichero.
Los lectores acceden a este fichero cada 50 milisegundos para leerlo, siempre y cuando se les conceda acceso a la sección crítica (acceso al fichero), esto se debe a que los escritores pueden modificar este fichero, y mientras esto ocurre ningún otro proceso podrá acceder al documento (no se les concederá el acceso a la sección crítica). Además las modificaciones que realice un escritor deberán quedar reflejadas en las copias de los ficheros de cada uno de los procesos.
Para ello se hará uso de las funciones que encontramos en el fichero gestorfichero.go, las cuales son: EscribirFichero, y LeerFichero.

Por último se ha implementado la función **Recibir** en el fichero ra.go, está función es por la cual se realiza el tratado del tipo de datos que son recibidos, para ello se ha implementado un switch el cual va a tratar los siguientes tipos de mensajes:

- **Request:** este caso es el que permite decidir qué proceso va a ser el que entre en la sección crítica. Para ello se comprueba si el proceso en el que se está es el que debería entrar a sección crítica o debería enviar su permiso a los demás procesos para que sean ellos quienes entren en la sección crítica.
- **Replay:** es el caso más sencillo, cuando se recibe un mensaje de este tipo significa que ha llegado un permiso al proceso que se está ejecutando y este se procesa mediante el canal crep.
- **RelojVector:** mediante este caso se puede recibir la información producida por el GoVector mediante la función **PrepareSend**, permitiendo recogerla y decodificarla con la función **UnpackReceive** para realizar así la correcta actualización de los relojes vectoriales.

Realización de pruebas

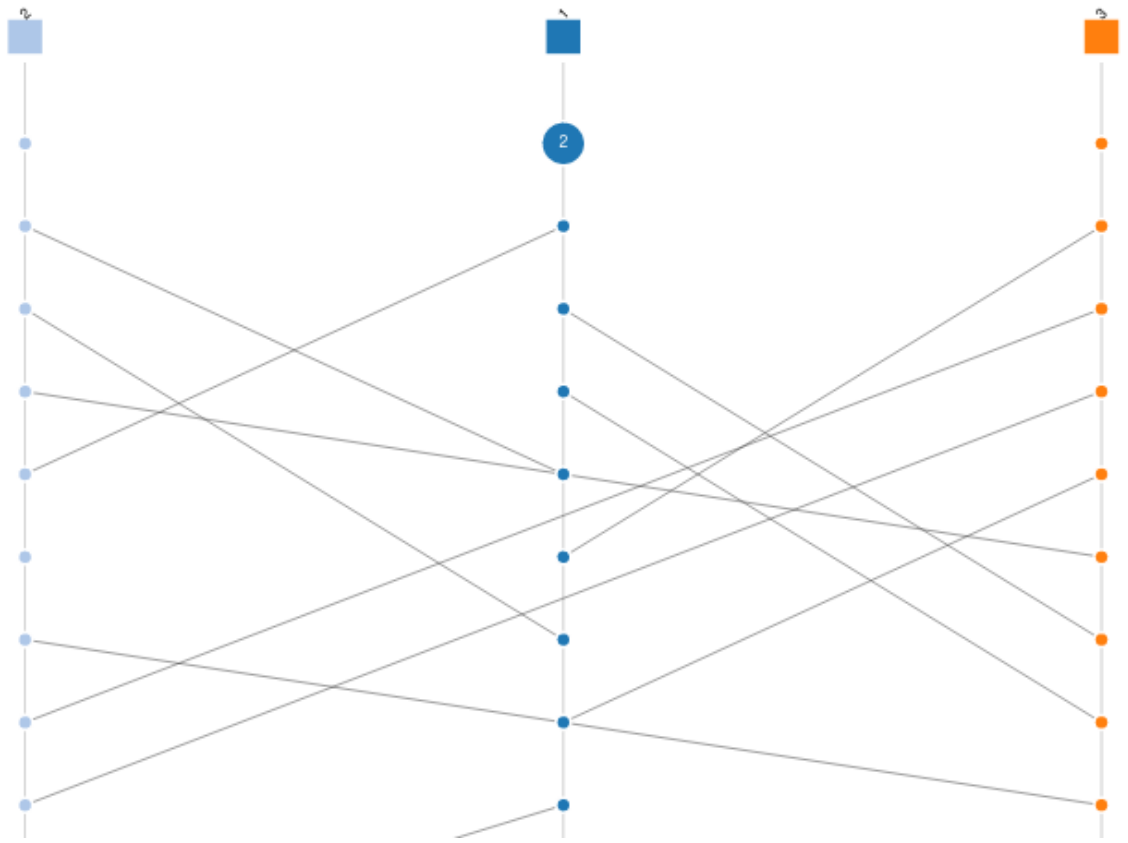
Para comprobar el correcto funcionamiento de este algoritmo aplicado al problema de los lectores, escritores, se ha hecho uso de las funciones proporcionadas por la librería GoVector, la misma que nos ha brindado las herramientas anteriormente para la gestión del vector en el que se guarda el valor más reciente recibido de los relojes de los procesos que se están ejecutando.

Para estas pruebas se ha generado un fichero Log para cada uno de los procesos, los cuales registraban todos los eventos que se enviaban, y que se recibían. Esto nos ha permitido estudiar el flujo de los mensajes enviados, y depurar el código, hasta corroborar su correcto funcionamiento.

En adición una vez se ha comprobado el funcionamiento de estos procesos se ha hecho uso de la herramienta shiviz, para el uso de la misma se han fusionado los documentos Lg de cada uno de los procesos, añadiendo al principio de este nuevo log, al cual le daremos el nombre de Shiviz.log.txt, establecemos la siguiente cabecera:

(?<host>\S*) (?<clock>{.*})\n(?<event>.*)

La cual permite que shiviz escanee el documento, encargándose de comprobar la correcta secuenciación de los mensajes establecidos en el Log. Una vez verificado el envío correcto de los mensajes se puede crear un diagrama de flujo de mensajes, en nuestro caso se ha seleccionado una prueba en la que se lanzaban dos procesos lectores, y un proceso escritor. Dado que se ha considerado que así se aprecia el intercambio de mensajes entre los diversos procesos.



Esta verificación se ha respaldado el estudio de los mensajes proporcionados por cada uno de los procesos, los cuales se volcaban en la terminal. Al igual que con comprobaciones manuales de los estados de los ficheros pertenecientes a cada uno de los procesos tras la ejecución de cada una de las pruebas.