Práctica 2

Memoria

Juan Antonio Rodríguez Gracia (NIP: 805001)
Miguel Beltran Pardos (NIP: 800616)

Sistemas distribuidos
Grado en Ingeniería Informática

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza
Curso 2021/2022

1. Introducción

Hemos procedido a diseñar y programar las soluciones a las diferentes archivos pedidos en la practica. Con el algoritmo de exclusión mutua de Ricart y Agrawala.

1. ra.go
2. lector.go
3. escritor.go
4. gestorfichero.go
5. memoria

2. Descripción de la implementación

Para la implementación del algoritmo hemos traducido la versión en algol[1] que evita condiciones de carrera. Para la implementación en lenguaje GO nos hemos apoyado en el código compartido por el profesor de tal modo que se ha utilizado el modelo actor a través de la clase ra.

Se ha añadido a la estructurada RASharedDB el Id del proceso que genera el Objeto ra, el tipo (lector o escritor) y un puntero a fichero para que el proceso edite su fichero con los cambios del resto de escritores. De tal manera que cuando un nuevo proceso llame al New para generar un objeto de tipo ra rellenara esta estructura la cual se usara en los diferentes procesos.

Una vez teníamos en la estructura completa para implementar el algoritmo en Algol, vimos que indica que es necesario crear 3 procesos 1 para el control de acceso a la SC, 1 para recibir request y actuar en consecuencia y otro para recibir replys.

En nuesta implementación se genera un proceso receivesMessages() con una gorutine el cual sera el encargado de procesar los diferentes mensajes que lleguen al Buzón de este proceso. Realiza la funcionalidad descrita en el algoritmo en algol de RECEIVES REQUEST", RECEIVES REPLY", además se le ha añadido una funcionalidad extra que recibe un mensaje de tipo token lo cual significa que debe escribir una nueva linea en su fichero.

Esto se ha implementado para poder tener los ficheros sincronizados ya que simulamos el acceso a un único fichero. Para realizar estas funcionalidades el proceso estará constantemente esperando mensajes de tal manera que una vez lo reciba evaluara el tipo del mensaje y según el tipo de mensaje realizara una funcionalidad u otra.

En el caso de que sea una REQUEST actualizara el reloj del proceso y comprobara las condiciones para ver si tiene que mandarle una REPLY al proceso o si lo introduce en la lista de avisar en postprotocol.

En caso de ser una REPLY decrementara el valor de OutRepCnt de tal manera que una vez reciba todas las REPLYS desbloqueara al proceso del chan para que pueda acceder a la SC.

En el caso de recibir TOKEN realizaremos la escritura en nuestro fichero.

Una vez realizado el proceso que gestiona los mensajes siguiendo la estructura del código suministrado por el profesor dividimos el proceso "WHICH INVOKES MUTUAL EXCLUSION FOR THIS NODE.[en] preprotocol y postprotocol.El preprotocol es el encargado de enviar una REQUEST a todos los procesos, esta REQUEST significa que quiere acceder en la SC una vez todos los procesos le hayan enviado su correspondiente REPLY, este podrá acceder a la SC. Esto se realiza a través de un chan de bool comentado anteriormente.

[1] Algol

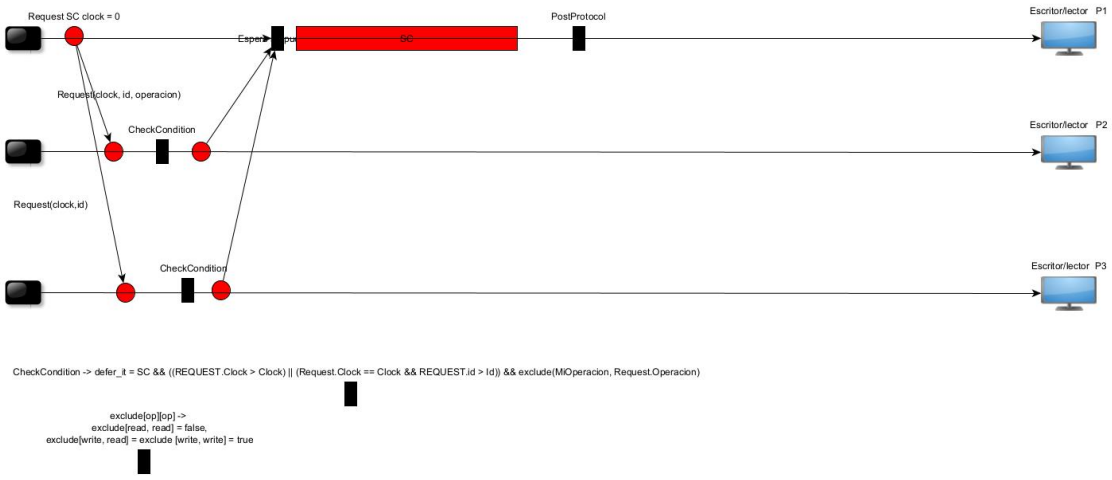
Una vez se acceda a la sección critica el proceso debe de llamar a postprotocol para comprobar si hay algún proceso a la espera de acceder a la sección critica.

También hemos añadido el método AccesSeccionCritica el cual es el encargado de enviar un mensaje(TOKEN) a el resto de procesos con los cambios que se han realizado en el fichero de tal manera que todos los procesos tendrán sincronizado el fichero.

3. Diagramas de secuencia

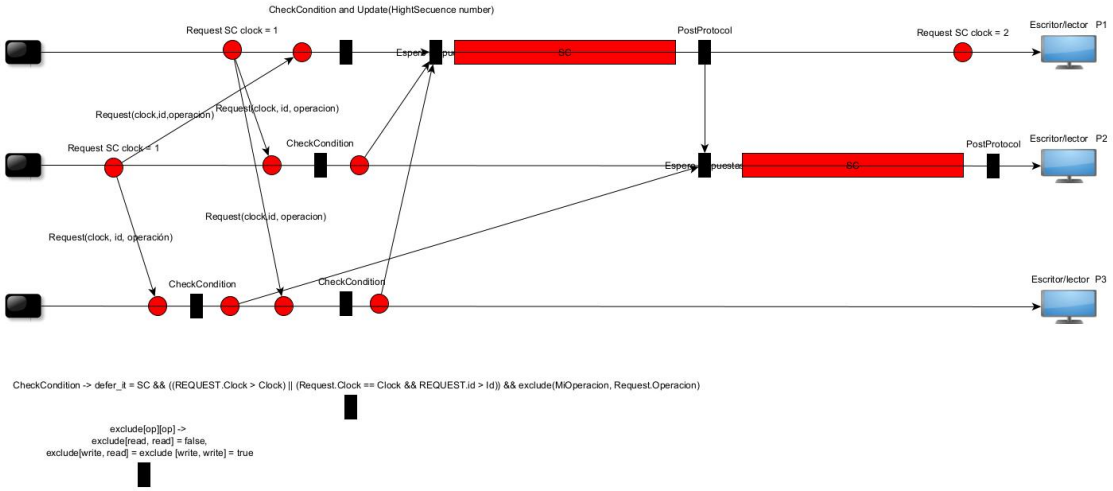
- **Caso de un solo nodo pidiendo acceso a la sección critica**

  Como se puede observar como el nodo P1 envía una REQUEST con su pid y su tiempo de reloj a N-1 nodos para pedir acceso a la sección critica. Los nodos P2 y P3 reciben esa petición (REQUEST), procesándola y comprobando si pueden responder en ese momento a la REQUEST o es necesario guardar la REQUEST para enviarle el REPLY posteriormente. En este caso, como solo un nodo pide sección critica dicha REQUEST seras respondida por los nodos al momento, donde el nodo P1 que requiere usar la sección critica estará esperando a que el resto de nodos le envíen el REPLY dándole permiso. Una vez hecho esto, P1 comenzara su sección critica y posteriormente realizara el PostProtocol donde se encarga de responder a todas las REQUEST que tenga almacenas del resto de nodos pidiendo sección critica, en este caso no tendrá ninguna.



- **Caso de varios nodos pidiendo acceso a la sección critica**

  En caso de tener varios nodos pidiendo sección critica, entrara aquel que tenga menor numero de reloj o el que tenga un numero de reloj igual a otro y además un pid menor. Cada nodo procesara las REQUEST que tiene y comprobara si puede atenderla al momento o guardarla para mas tarde. Como podemos ver en la gráfica, el nodo P1 y P2 piden al resto de nodos acceso a la sección critica. P2 envía a P3 y P3 le envía el REPLY al momento, a diferencia de P1 ya que también había solicitado la sección critica. Por lo que P2 se queda a la espera de P1, mientras que P1 recibe el REPLY de P3 y de P2, teniendo en cuenta que los relojes tienen el mismo valor y que el pid de P1 sera menor que el de P2 por lo que tendrá prioridad P1. Debido a esto P1 ejecutara su sección critica y una vez terminada, en el PostProtocol enviara las REPLY que tenia guardadas a los nodos correspondientes, en este caso a P2. Una vez que P2 reciba la REPLY de P1, comenzara su sección critica.



4. Comprobación del sistema

4.1. Desarrollo

Durante la implementación del código se realizaron test de forma unitaria de tal forma que testeamos las diferentes funciones que implementábamos para comprobar que eran correctas.

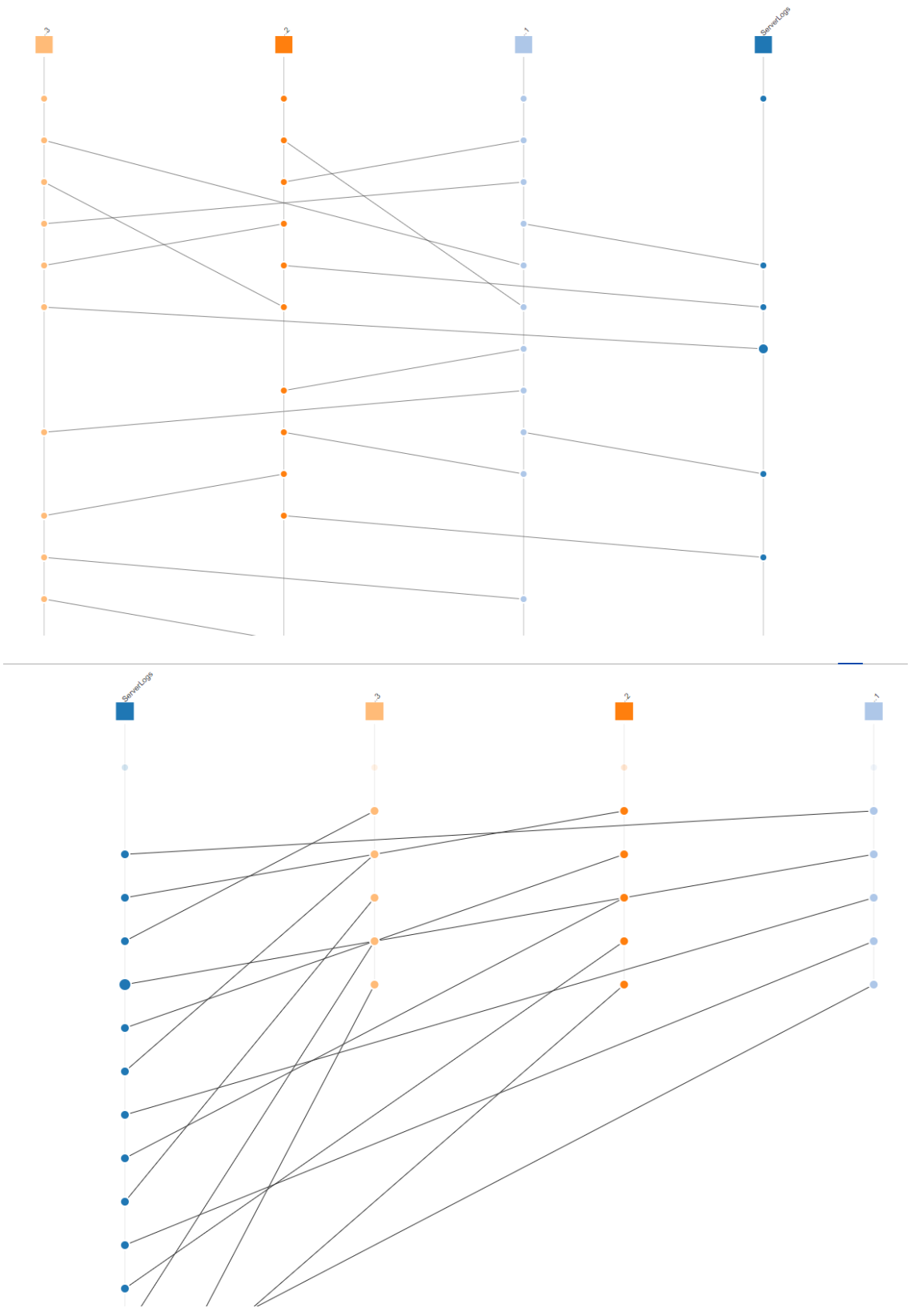
Una vez desarrollamos el código completo procedimos a realizar la implementación de GoVector en cual nos ha permitido comprobar de una manera visual que nuestro algoritmo estaba implementado correctamente. Para ello hemos tenido que simular un fichero remoto al cual acceden los diferentes escritores y lectores. Para los diferentes experimentos con GoVector se ha realizado un PrepareSend() a la hora llamar al preprotocol. Una vez se accede a la SC se conecta con el fichero remoto y se realiza un UnpackReceive(). Mas adelnate añadimos los diagramas donde puede apreciarse el funcionamiento.

También se ha comprobado a través del comando diff de linux que los diferentes ficheros contiene el mismo contenido.

4.2. Experimentos

4.2.1. Prueba con 3 escritores simultáneos

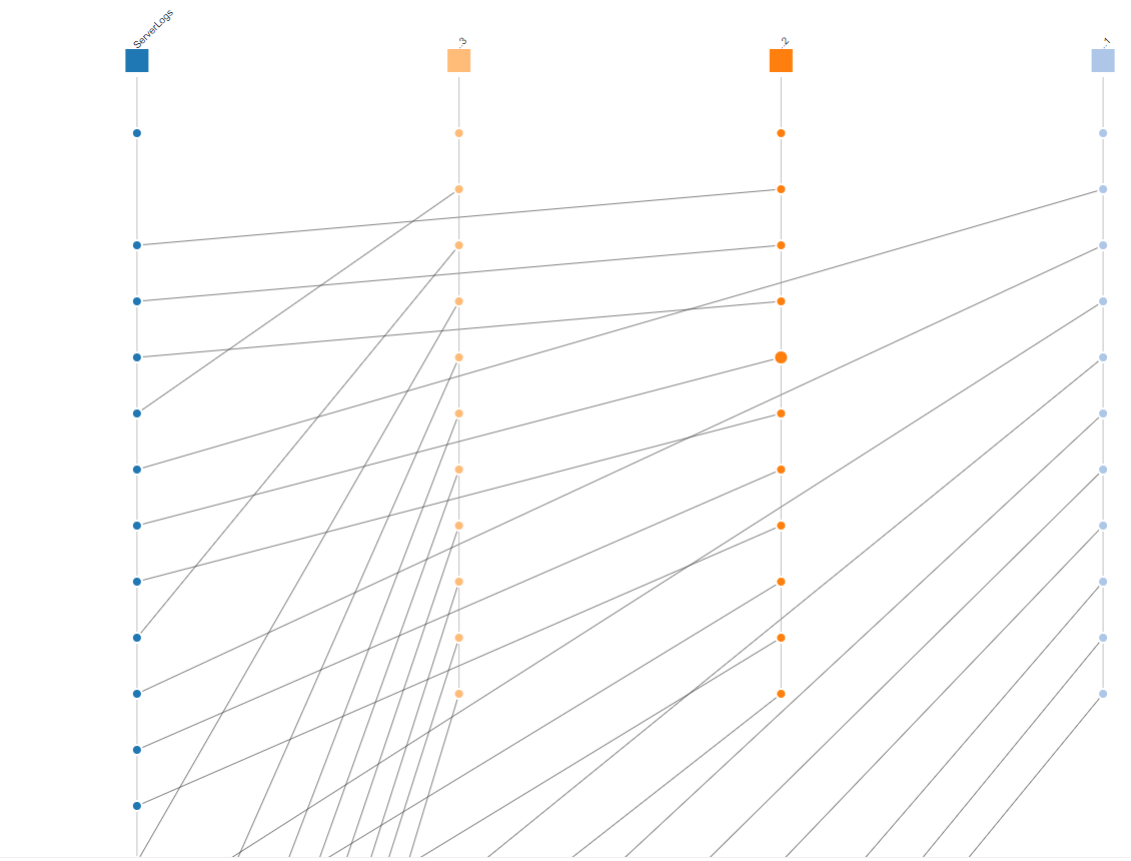
Se ha ejecutado simultáneamente 3 procesos escritores realizando 5 escrituras simultaneas. De tal forma que puede apreciarse como los diferentes procesos acceden según si el reloj es menor y su identificador es menor al resto.



4.2.2. Prueba con 3 escritores con diferente tiempo de acceso a la SC

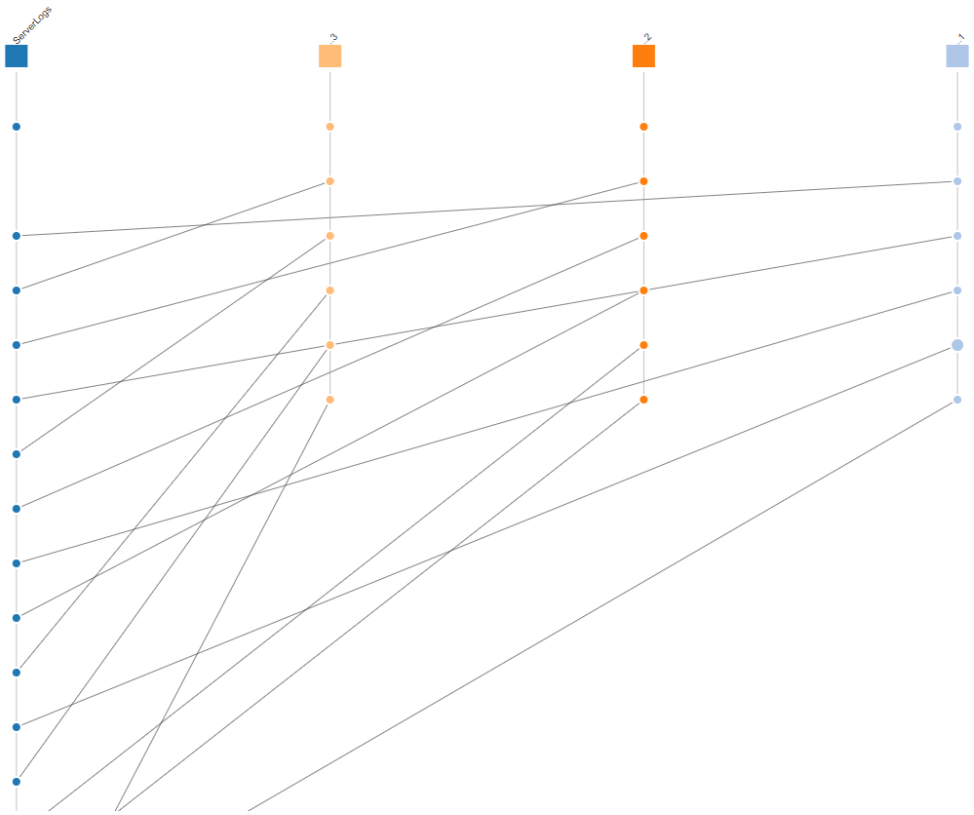
Se han ejecutado simultaneamente 3 procesos escritores realizando 5 escrituras simultaneas de tal forma que el proceso 1 realiza escrituras cada 5 segundos, el proceso 2 realiza escrituras cada segundo y el proceso 3 cada 4 segundos. Como se puede apreciar en la gráfica el proceso 2 accede a la sección critica hasta que el

proceso 3 quiere acceder a ella de tal modo que se actualizan los relojes y accede el proceso 3



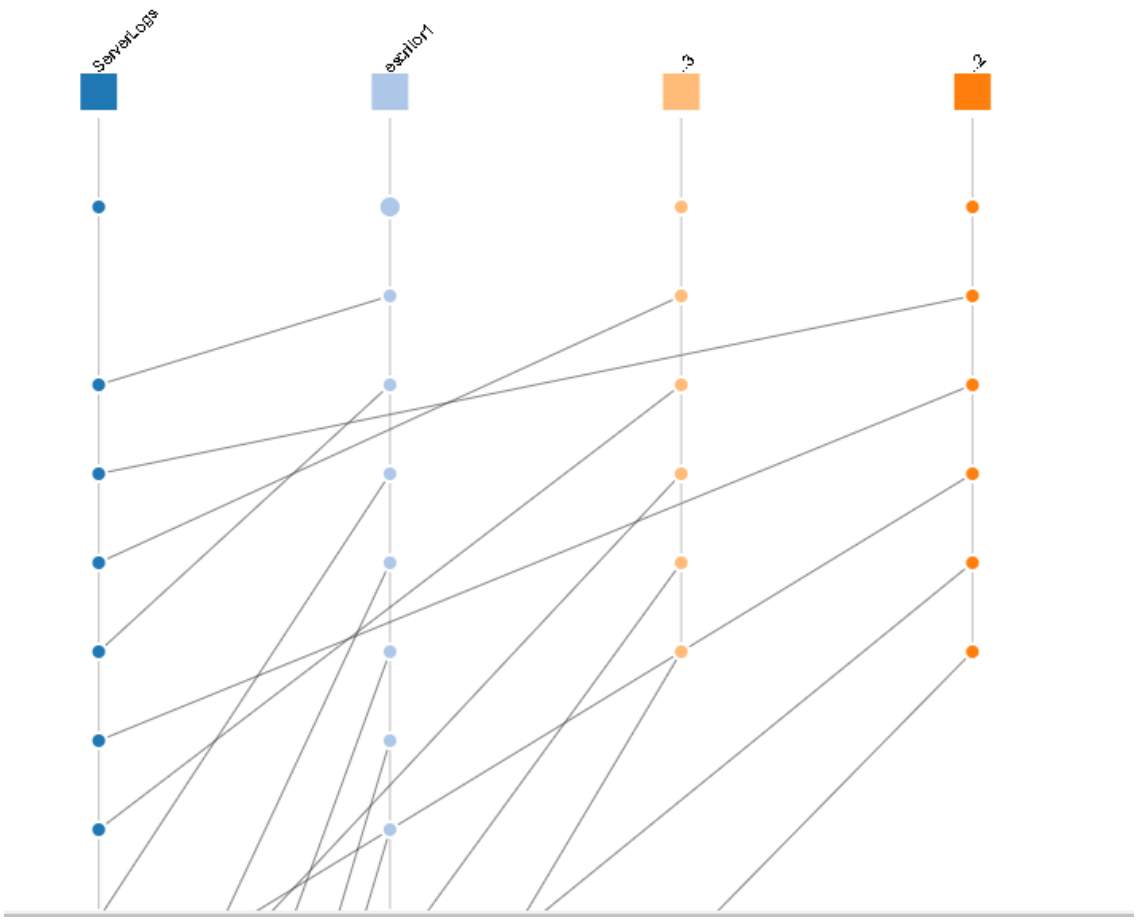
4.2.3. Prueba con 3 lectores

Se han ejecutado simultáneamente 3 procesos lectores realizando 5 lecturas simultaneas.



4.2.4. Prueba con 1 escritor y 2 lectores

Se han ejecutado 2 procesos lectores y 1 escritor, donde el escritor ha realizado escrituras cada 3 segundos, y un lector ha realizado lecturas cada 1 segundo y el otro cada 5 segundos. En la siguiente figura donde la segunda entidad es el escritor y los dos siguientes son los lectores, podemos observar que van accediendo a la SC según indica el algoritmo de tal forma que los procesos lectores podrán leer simultáneamente hasta que el proceso que quiere escribir acceda a la SC


ServerLogs
escritor1
...3
...2