SISTEMAS DISTRIBUIDOS

PRÁCTICA 2

874055 - Ariana Porroche Llorén
871627 - Irene Pascual Albericio

ÍNDICE

Introducción

Este programa consiste en un sistema distribuido donde para acceder al recurso compartido (zona de sección crítica), deben coordinarse. La coordinación la hemos llevado a cabo mediante el algoritmo de Ricart & Agrawala modificado con los relojes vectoriales.

El sistema utiliza múltiples nodos que actúan como lectores y escritores, donde cada uno va interactuando (los primeros escriben en el archivo, y los segundos leen lo modificado en el archivo).

El directorio cmd, contiene 2 subdirectorios: lector y escritor, cada uno con su propio código. Aquí se puede comprobar cómo los escritores solicitan acceso a la sección crítica cuando van a hacer la tarea relacionada con el documento compartido, y notifican su salida una vez hayan finalizado su tarea. En cambio, los lectores pueden acceder a la vez, ya que no son operaciones excluyentes. Los tipos de operaciones tienen definidos si son excluyentes o no en el archivo lectorWriter.go.

En el directorio ms, el archivo ms.go se encarga de gestionar la comunicación entre los nodos (tanto lectores como escritores): envía solicitudes, recibe solicitudes, maneja respuestas, coordina el acceso… todo esto se hace posible gracias al canal listener.

Por último, en el directorio ra, el código del archivo raVect.go contiene la lógica específica para el algoritmo mencionado anteriormente. Aquí se hace uso de dos vectores importantes: OurSeqNum (en el código aparece como el vector ldr) y HigSeqNum (que se corresponde con el vector clk).

- OurSeqNum representa el número de secuencia de la última solicitud de acceso a la sección crítica por un proceso. Se incrementa cada vez que el proceso envía una nueva solicitud de acceso.
- HigSeqNum representa el número de secuencia más alto que el nodo ha visto hasta ese momento (tanto de sus propias solicitudes como de las recibidas de otros nodos). Se actualiza cogiendo el valor máximo entre su valor actual del HigSeqNum y el valor recibido de otro nodo.

Gracias a esta arquitectura, se garantiza que el acceso al recurso compartido sea ordenado y sincronizado, evitando conflictos y asegurando que todas las solicitudes sean atendidas justa y consistentemente.

Ejercicio 1

1. Diseño del programa

Algunos de los aspectos más importantes del código:

- El tipo cmd.lectorWriter de la función New de raVector.go, toma el valor que le pasan del escritor/lector y lo guarda en el campo lw de RASharedDB. Cuando quiera enviar una petición, se enviará con ese campo y se guardará en el campo de Request.
  Cuando llegue una petición, la operación Exclude del archivo lectorWriter.go establecerá si son excluyentes o no. Si lo son, se comprobará cuál debe entrar antes a la sección crítica (entrará primero el que menor reloj tiene, y si son iguales, el que tenga menor identificador de proceso). Si no son excluyentes, podrán entrar a la vez.
- El canal listener permite a los nodos estar atentos a las solicitudes y respuestas que se envían entre sí. Cada nodo tiene su propio listener.
- Perm_delayed (RepDefd) es una estructura que contiene información sobre los nodos que han sido puestos en la cola de espera. Por ejemplo, si llega una solicitud mientras está el proceso está la sección crítica o si el proceso tiene más prioridad que el solicitante, pondrá al proceso que solicitó en espera.
  Cuando el nodo salga de la sección crítica, revisará los procesos en espera y enviará los permisos pendientes a los nodos que solicitaron.
- Waiting_from es una estructura que contiene la información sobre los nodos de los cuales, el nodo actual, está esperando su respuesta. Esto se utiliza cuando el proceso solicita entrar en la sección crítica, ya que debe esperar a los N-1 nodos a que le den permiso para que pase.
  Si algún nodo no le da permiso, se quedará esperando hasta recibirlo.
- Los relojes vectoriales son actualizados cuando se reciben solicitudes o respuestas. Cada nodo tiene sus propios relojes, pero cuando reciban mensajes de otros nodos, deberá sincronizar el suyo con el de los demás.

Este es el algoritmo de Ricart & Agrawala que hemos seguido, haciendo uso del código Algol proporcionado, y de las modificaciones con los relojes vectoriales:

PreProtocol()

```
P(Shared_vars);
  (1)   Requesting_Critical_Section := TRUE;
  (2)   Highest_Sequence_Number[myPid]++
        Our_Sequence_Number[myPid] := Highest_Sequence_Number[myPid]
V(Shared_vars)
  (3)   Outstanding_Reply_Count := N - 1;
  (4)   for j := 1 STEP 1 UNTIL N DO
            IF j != me THEN
                Send_Message(REQUEST(Our_Sequence_Number[], me), j);
  (5)   WAITFOR (Outstanding_Reply_Count = 0);
```

PostProtocol()

```
  (7)   Requesting_Critical_Section := FALSE;
  (8)   FOR j := 1 STEP 1 UNTIL N DO
        IF Reply_Deferred[j] THEN
            Reply_Deferred[j] := FALSE;
            Send_Message(REPLY, j);
```

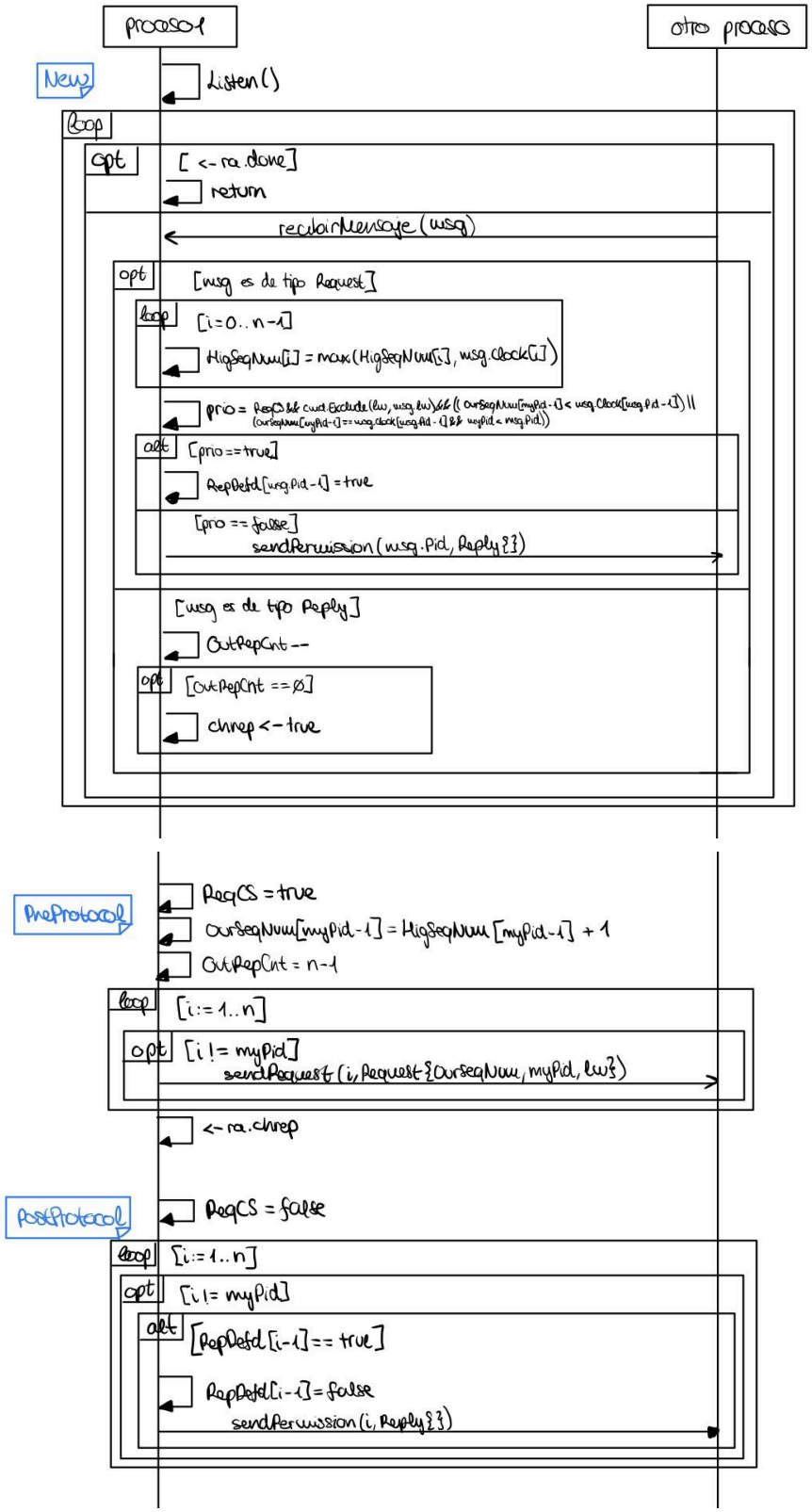
RequestReceived(k, j)

```
  (10)  FOR i := 1 STEP 1 UNTIL N DO
            Highest_Sequence_Number[i] := Max(Highest_Sequence_Number[i], k[i]);
P(Shared_vars);
  (11)  Defer_it := Requesting_Critical_Section AND ((k[] > Our_Sequence_Number[])
                                OR (k[] = Our_Sequence_Number[] AND j > me));
V(Shared_vars);
  (12)  IF Defer_it THEN
            Reply_Deferred[j] := TRUE;
  (13)  ELSE
            Send_Message(REPLY, j);
```

PermissionReceived(j)

```
  (15)  Outstanding_Reply_Count := Outstanding_Reply_Count - 1;
```

2. Diagramas de secuencia



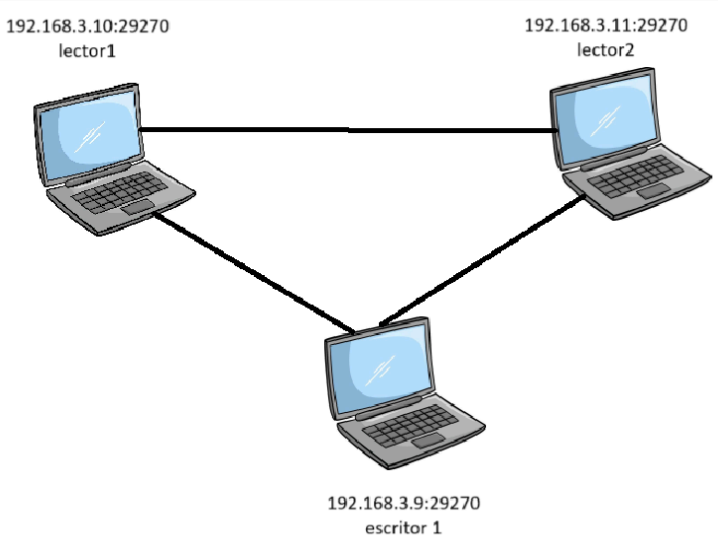
3. Máquinas de estado

Hemos evaluado el código en diversas configuraciones. Todas están probadas con el fichero de texto “fichero.txt”, en el cual leen y escriben todos los procesos. Y en “ficheroParticipantes.txt” definimos las ips y los puertos de las máquinas con las que vamos a hacer las pruebas, en nuestro caso, el contenido es el siguiente:

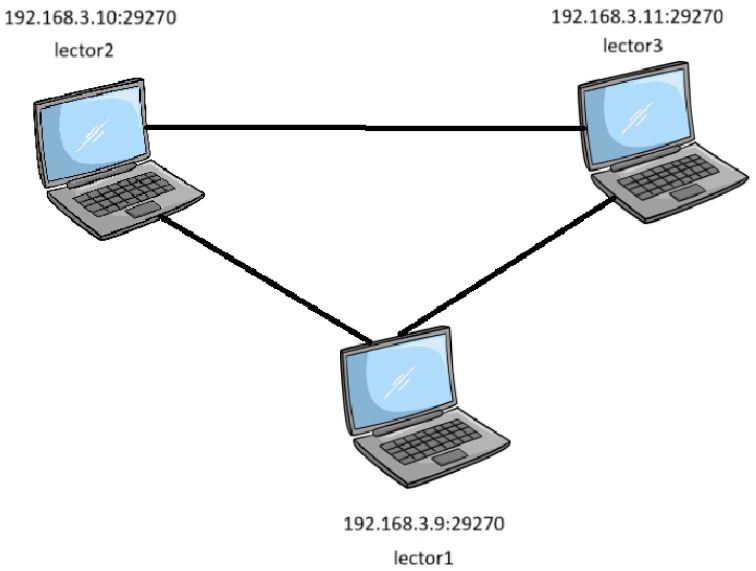
192.168.3.9:29270
192.168.3.10:29270
192.168.3.11:29270

A continuación mostraremos algunos ejemplos probados:

1 escritor y 2 lectores: el escritor accede a la sección crítica, sin colisionar con los lectores. Una vez termine su tarea, notificará su salida de esta sección, y los dos lectores procederán a leer (a la vez o no) el fichero modificado.



3 lectores: el programa hará que todos los nodos accedan al fichero compartido (pudiendo acceder a la vez si se da la ocasión).



2 escritores y 1 lector: en este caso, todos los nodos deberán actuar cuando reciban los permisos del resto, ya que los escritores no pueden acceder a la vez, y el lector tampoco puede acceder a la vez que el resto. Entonces esto es un ejemplo donde todas las operaciones vistas son excluyentes.