

Variante algoritmo exclusión mutua

Román Larrosa Lewandowska

1. Pseudocódigo

a. Variables utilizadas

```
type prioridad = enum{ALTA, BAJA};  
n: int           //Número de clientes  
m: int           //Número de réplicas  
replica: int     //Réplica asignada a un cliente  
pr: prioridad    //Prioridad asignada a un cliente
```

b. Canales de comunicación

```
chan token [1:m]();  
chan solicitar [1:m]();  
chan conceder [1:n]();  
chan liberar [1:m]();
```

c. Clientes

```
Cliente[i:1..n](replica:1..m, pr:prioridad)::  
  do true →  
    //...  
    /*Enviamos solicitud a réplica asignada al cliente,  
    *indicando el número de cliente y la prioridad */  
    send solicitar[replica](i, pr);  
    receive conceder[i]();  
    //Sección crítica  
    send liberar[replica]();  
    //Sección no crítica  
    //...  
  end
```

d. Réplicas

```
Replica[j:1..m]()::  
  var HP: queue of int; //Cola de alta prioridad (High)  
  var LP: queue of int; //Cola de baja prioridad (Low)  
  
  do true →  
    //Si hay alguna petición...  
    if not (empty(solicitar[j])) →  
      receive solicitar[j](id, prioridad);
```

```

        if prioridad == ALTA →
            HP.insert(id);
        fi
        if prioridad == BAJA →
            LP.insert(id);
        fi
    fi
    //Si hay token que recibir...
    if not (empty(token[j])) →
        receive token[j]();
        //Primero: procesos de alta prioridad
        do not empty(HP) →
            id = HP.remove();
            send conceder[id]();
            receive liberar[j]();
        end
        //Procesos de baja prioridad
        do not empty(LP) →
            id = LP.remove();
            send conceder[id]();
            receive liberar[j]();
        end
        //sleep ??

        //Pasamos el token a la siguiente réplica
        send token[(j mod m) + 1]();
    fi
end

```

2. Explicación

En la solución aportada, se han pretendido abordar los siguientes problemas:

- Más de un cliente por réplica.
- Clientes tienen distinto nivel de prioridad para entrar a la sección crítica de su código.
- Aproximación a la relación causal Ocurrió-Antes

Para ello, el primer cambio se produce en los clientes. A la hora de realizar la solicitud a una réplica, estos deben pasarle a la réplica su número de proceso y la prioridad. En la solución aportada en las transparencias no es necesario puesto que únicamente hay un cliente por cada réplica, pero ahora hay que enviar al canal solicitar de la réplica asignada al cliente estos datos. Por parte del cliente ya no hay más cambios.

Los canales de comunicación sufren una pequeña modificación:

- El canal solicitar, pasa a tener dos argumentos de entrada: el cliente que manda la solicitud, y la prioridad del mismo.
- Conceder pasa a tener tantos canales como clientes haya.
- Token y Liberar mantienen el número de canales: tantos como réplicas

En el caso de las réplicas hay bastantes más cambios. El primero es añadir dos colas a cada proceso, una para los clientes con prioridad alta y otra para los clientes cuya prioridad sea baja. Cada réplica ejecuta un bucle infinito en el que:

1. Si hay alguna solicitud, ésta se recibe, y se añade el id del proceso cliente a la cola correspondiente dependiendo de su prioridad.
2. Si el canal por el que se recibe el token no está vacío, se recibe el token, y a continuación:
 - a. Se concede acceso por orden de llegada de las peticiones a los clientes con prioridad alta (por cada uno se envía la concesión y se espera por la liberación de la réplica).
 - b. Se concede acceso por orden de llegada de las peticiones a los clientes con prioridad baja (idem).
 - c. Se envía el token a la siguiente réplica.

El cumplimiento de la relación causal Ocurrió-Antes es parcial, puesto que con ésta variación del código, se garantiza que las peticiones realizadas a cada réplica se ejecuten de manera ordenada, sin embargo no se puede garantizar que ocurra así entre los clientes de todas las réplicas, puesto que el algoritmo del anillo no tiene en cuenta dicha relación. Mediante las guardas es imposible que ningún proceso pueda entrar en inanición, y si suponemos los canales de comunicación fiables, ésta implementación es segura puesto que únicamente un proceso está accediendo a la vez a su sección crítica

3. Referencias

- a. Distributed Systems Lecture
<http://tele.informatik.uni-freiburg.de/Teaching/ws01/dsys/dsys.html>, última visita 5 abril 2020.
- b. Lenguaje de Comandos Guardados - Wikipedia
https://es.wikipedia.org/wiki/Lenguaje_de_Comandos_Guardados, última visita 5 abril, 2020.
- c. Linux Programmer's Manual