

ASRR SCD

Problema de sección crítica

1 - Introducción histórica

- Cerrajos
- Desde 1965, soluciones software indep. de una máquina concreta
- Proporcionan acceso equitativo
- Ahora necesario encontrar versiones descentralizadas de alg. clásicos
- Tipos → Centralizados → Van compartidas
Distribuidos → Pasan mensajes

2 - Solución con espera ocupada

- Procesos esperan a la entrada de la sección crítica
- Solución aceptable si no hay muchos procesos
- Solución con salto interno para 2 procesos pero no cumple cond. de seguridad.

C...?) Hacemos alg. que tema 2

9 - Problema de exclusión mutua en un entorno distribuido

- Hipótesis previas
 - Red completamente conectada
 - Transmisión de msg. sin errores
 - Retraso variable pero acotado
 - Pueden darse desencuadramientos

9.1 Alg. Original de Ricart-Agrawala (1981)

- Se propone para reducir el nº de mensajes necesario
- Necesita $2(n-1)$ mensajes / proceso con n procesos
- Pasos:

- 1- P_i quiere entrar, genera número de secuencia ns y difunde petición req.
 - 2- P_j recibe req y puede aceptar → manda req o pospone su respuesta
 - 3- P_i entrará en la SC cuando tenga respuesta favorable de todos
 - 4- Cuando salga, responderá fav a qn esté esperando
- Regla de prioridad: si P_i no quiere entrar responde fav. Si quiere entrar, compara su ns con el de la ultima pet recibida, si el suyo es más reciente, pospone su respuesta

```

Version Concurrente del algoritmo de Agrawala
var
ns: 0..+INF; (* numero de secuencia generado por P(i) *)
mns: 0..INF:= 0; (* mayor numero de secuencia generado o recibido *)
numrepesperadas:0..n-1;
intentaSC: boolean;
prioridad: boolean;
repretrasadas: array[1..N] of boolean;

Proceso Main(i)
(1) wait(s);
(2)   intentaSC:= TRUE;
(3)   ns:= mns +1;
(4)   numrepesperadas:=n-1;
(1) signal(s);
(6) for j:= 1 to n do
(7)   if j <> i then
(8)     send(j, pet, ns, i);
(9) wait(sinc);
<<sección critica>>
(10) wait(s);
(11)   intentaSC:= FALSE;
(12) signal(s);
(13) for j:=1 to n do
(14) if repretrasadas[j] then begin
(15)   repretrasadas[j]:= FALSE;
(16)   send(j, rep);
end;

```

```

Proceso Pet(i)
(1) receive(pet, k, j);
(2) wait(s);
(3)   mns:= max(mns, k);
(4)   prioridad:= (intentaSC) and (k>ns or (k=ns and i<j));
(5)   if prioridad then
(6)     repretrasadas[j]:= TRUE
(7)   else send(j, rep);
(8)   signal(s);

Proceso Rep(i)
(1) receive(rep);
(2)   numrepesperadas:= numrepesperadas -1;
(3)   if numrepesperadas= 0 then (* el proceso puede entrar en S.C.*)
(4)     signal(sinc);

```

9.2 - Demostración de corrección

Dado P_i y P_j siempre se cumple que uno ha de entrar antes que otro

Por reducción al absurdo: si los dos entran a la vez \Rightarrow han enviado pet al otro y han recibido resp del otro. Casos:

1 - $P_i \xrightarrow{\text{rep}} P_j$ antes de asignar su nro de sec. Pero $ns_j > ns_i \Rightarrow P_j$ responde contestar. Dújelo contradicción.

2 - $P_j \xrightarrow{\text{rep}} P_i$ antes ... Análogo

3 - $P_i \xrightarrow{\text{rep}} P_j \wedge P_j \xrightarrow{\text{rep}} P_i$ Ambos con intentaSC=true. Solo 1 retrasado por los nros de sec. En caso de ser iguales, entra el que tenga < n° de nodo. La propiedad de alcanzabilidad lo pone recto al absurdo. Sup que ninguno entra por esperar una respuesta favorable de otros proceso \Rightarrow Retraso indef de la respuesta incompatible por la condición de prioridad. Siempre IPi con menor ns que recibe todas las contestaciones favorables

La propiedad de ausencia de maniobras por razonamiento similar.

Como las peticiones están globalmente ordenadas por antigüedad \Rightarrow todas reciben resp fav en algún momento

Propiedad de equidad: según la ejecución más desfavorable y considerando que los resp pueden adelantarse a pet el número de turnos será $d_1(c_n) = \frac{n(n+1)}{2} - 1$ ya que P_i puede adelantarse a P_j como máx $ns_j - ns_i$ turnos. En caso de que resp NO pueda adelantarse a pet $\Rightarrow P_i$ no puede adelantarse y entrar más de 2 veces seguidas en SC (1 por ns, 2 por n° de nodo). Entonces el máx retraso será $d_2(c_n) = 2(n-1)$

9.3 Alg. de Suzuki-Kasami (1982) / Ricart-Agrawala (1983)

- Permisos para entrar = posesión de un token
- Token se asigna arbitrariamente al principio.
- Los demás no saben quién lo tiene
- Lo solicitan por difusión → msj con su ns
- Token = array cuyo k-ésima elem = ns de Pk de su ret adquisición
- Una vez Pk sale de la SC, busca según j+1 n, i-1 en un array local de peticiones hasta encontrar hecha después de la última vez que lo obtuvo
- Si no hay, vuelve a entrar a SC

Versión concurrente del algoritmo:
 token_presente:boolean:=FALSE;
 enSC:boolean:= FALSE;
 token: array[1..N] of 0..+INF;
 petición: array[1..N] of 0..+INF;

Proceso Pi
 (0) wait(s);
 (1) if NOT token_presente then begin
 (2) ns := ns+1;
 (3) broadcast(pet, ns, i);
 (4) receive(acceso, token);
 (5) token_presente := true;
 (6) end;
 (7) enSC := true;
 (8) signal(s);
 <<sección crítica>>
 (9) token[i] := ns;
 (10) enSC := false;
 (11) wait(s);
 (12) for j := i+1 to n, 1 to i-1 do
 (13) if petición[j] > token[j] and token_presente then begin
 (14) token_presente := false;
 (15) send(j, acceso, token);
 (16) end;
 (17) signal(s);

Proceso Pet(i)
 (1) receive(pet, k, j);
 (2) petición[j]:= max(petición[j], k);
 (3) wait(s);
 (4) if token_presente and NOT enSC then
 <<< repetir (12)- (16) >>>
 signal(s);

9.4 - Demostración de la corrección del algoritmo

Demostrar ex. mutua \equiv en cualquier momento el nº máx de var token_presente=true es 1. Como inicialmente es uno, dem. que se mantiene en la transmisión

- Prot de adquisición: cambia false \rightarrow true cuando recibe token
- Prot de restitución: si envía \Rightarrow lo poseía \Rightarrow después cambia true \rightarrow false

Alcanzabilidad de SC sup que todos quieren entrar a SC y ninguno puede \Rightarrow no token. Es decir, token_presente=false $\forall i \Rightarrow$ token en tránsito por la red \Rightarrow En algún momento llega luego contradicción

Equivocidad: por la hipótesis de que los msj llegan en un plazo finito. Como se entrega el token al que lo haya pedido en orden $i+1, \dots, n, 1, \dots, i-1$ y no se pierden msj, eventualmente cualquier proceso entrará

Se necesitarán n mensajes / proceso \rightarrow n-1 solicitudes \rightarrow tipo, ns, id Proc. \downarrow
 1 respuesta $O(n^2)$ \rightarrow tipo, $n+1$ elem \rightarrow array token
 Es mejor que Ricarta-Agrawala (1981)

- Si la red no es fiable, sigue funcionando pero puede darse
- Finalización de algunos proc
 - Interbloqueo si se pierde el token