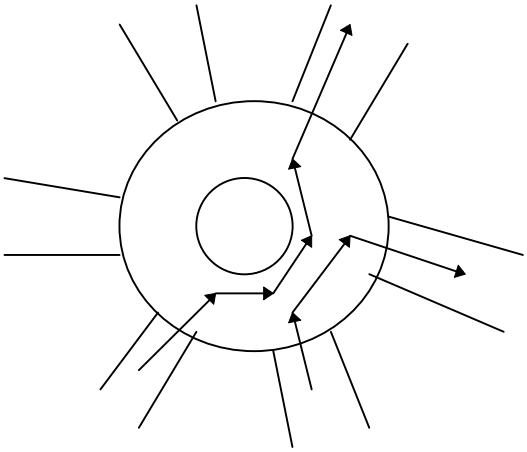
# Esercizio di Sincronizzazione Tra Processi: La Rotonda

## Testo:

Si consideri una rotatoria nel mezzo di una confluenza tra un numero N (per esempio 6) di strade. Secondo il vigente codice stradale

- il verso di rotazione è antiorario;
- un'auto ha la precedenza all'ingresso della rotonda e, una volta nella rotonda, deve dare la precedenza alle auto che vogliono accedere dai rami intermedi.



Poiché la rotatoria ha un ovvio limite fisico di capacità (il numero massimo di auto contenute è CMAX) è possibile una situazione di deadlock: la rotonda è piena di auto che non possono uscire perché in attesa di fare entrare auto dai lati intermedi, le auto dai lati intermedi non possono d'altronde entrare per problemi di capacità.
Si implementi una politica di gestione della rotonda, facendo uso del costrutto Monitor, che eviti situazioni di deadlock. Descrivere la struttura dei processi macchine. Discutere eventuali soluzioni dotate di maggiore parallelismo e i loro problemi.

## Soluzione:

{Per quanto riguarda la struttura della auto, un'auto generica deve:
accedere dal proprio ramo di ingresso solo se il deadlock non è possibile;
ruotare all'interno della rotonda dando la precedenza ad ogni ramo intermedio ma solo se questo non orovoca il deadlock. Ne deriva la seguente struttura per i processi macchina:}

```
program rotatoriaconcapacitamassima;
const N = numerorami ; { per esempio 6 }
type  rami = 1 .. N ;

type  auto = process;
var i,o: rami
{sono i rami di ingresso e uscita delle macchine}
begin
    repeat
        ROT. INGRESSO ( i );
        ROT. RUOTA ( i, o);
        ROT. ESCI ( o );
    until false;
end;

type  rotonda = monitor;
const NMAX = ..; { numero massimo di auto possibili }
var       cont : 0 .. NMAX ;
         enter , daiprec : array [ rami ] of condition;
{l'array di condition enter consente la sospensione delle auto all'esterno della rotonda  quando questa è piena - l'array daiprec consente la sospensione dei processi già  in rotazione rispetto a quelli in ingresso nella  rotonda che devono avere la precedenza }
         content : array [rami ] of integer;
            {contatore  delle  auto già entrate nel ramo  ma  non ancora transitate ed entrate in rotonda }

procedure entry INGRESSO (i : rami );
begin
    if cont = NMAX then enter [i].wait; { se la rotonda e' piena, sospensione}
    cont + := 1;  content [i] + := 1;
end;

procedure entry RUOTA  (i,o: rami);
var j : rami;
begin
    content [i] - := 1;
    if content [i]= 0 then
        while daiprec[i].queue do daiprec[i].signal;
{ vengono segnalati tutti i processi che  si sono sospesi e hanno dato la precedenza alle macchine in transito }
    for j := succ(i) to prec(o) do
       {per ogni ramo intermedio della rotonda si da la precedenza ai rami attraversati}
        while cont< NMAX and content[j] <> 0  do
                daiprec [j]. wait;
end;
```

```
function cequalcunoincoda : boolean;
var j:rami;
begin
      cequalcunoincoda := false;
      for j:= 1 to N   do
        if enter [j].queue then cequalcunoincoda := true;
end;
{la funzione succ e prec hanno il significato di somma e differenza modulo N }

procedure entry ESCI ( o: rami );
var j: rami ;
begin     cont - := 1;
         while cont < NMAX and  cequalcunoincoda do
           for j := 1 to Ndo
              while cont < NMAX and enter[j].queue do
                  enter [j].signal;
end;

begin
    cont := 0; for j in rami do content [j] := 0;
end;
var ROT : rotonda;
   pr,.....  : auto;
begin end.
```

{La starvation non è bandita da questa soluzione. Infatti la riattivazione dei processi esterni bloccati è sempre nello stesso ordine e quindi chi entra dai rami cn numero più basso sarà sempre favorito.
Si può pensare a politiche di riattivazione più giuste (in inglese FAIR e FAIRNESS)
Ad esempio:}

```
 j := succ ( o );
 while  cont < NMAX and cequalcunoincoda   do
    begin enter [j]. signal ;
         j := succ (j);
    end;
```

{In questo modo la riattivazione avviene sempre a partire dai rami successivi a quelli di uscita o per esempio a quelli di ingresso. Ancora, si può pensare a mantenere nel monitor una variabile che viene aggiornata ogni volta che una macchina esce e permetta di scorrere il risveglio circolarmente sui rami della rotatoria }

{Per quanto riguarda il maggiore parallelismo, si noti che:

1) si può pensare ad una struttura con un monitor per ogni ramo garantendo quindi che l'accesso ad un ramo non comporti il blocco di tutti gli altri rami. Ogni monitor mette a disposizione tre procedure: entra, esce e ruota. E' necessario considerare anche un monitor come arbitro. Deve infatti esistere un gestore globale che sia capace di tenere conto della condizione globale (cioè il numoer totale di auto presenti nella rotonda.
2) I monitor pero' sono in stretta interrelazione tra loro: i monitor dei rami devono poter chiamare quelli della procedura dell'arbitro per conoscere la situazione globale della rotonda, per poterla variare. Questo non comporta scorrettezze perché il monitor arbitro non invoca le procedure dei singoli rami, ciò fa in modo che non si introducano problematiche di deadlock;
3) I monitor dei vari rami, però, devono potersi segnalare tra loro, e questo reintroduce il problema del deadlock.