

Esercizio 3 (nell'esercitazione finale)

Considerate il seguente algoritmo di Dijkstra "modificato":

```
Initially
    /* global info */
    boolean interested[N] = {false, ..., false};
    boolean passed[N] = {false, ..., false};
    int k = <any> ;          // k ∈ {0, 1, ..., N-1}
    /* local info */
    int i = <entity ID>; // i ∈ {0, 1, ..., N-1}
repeat
    1  <Non Critical Section>;
    2  interested[i] = true;
    3  while (k != i) {
    4      passed[i] = false;
    5      if (interested[k]) then k = i;
    6  }
    7  passed[i] = true;
    8  for j in 0 ... N-1 except i do
    9      if (passed[j]) then goto 3;
    10 <critical section>;
    11 passed[i] = false; interested[i] = false;
forever
```

L'algoritmo di Dijkstra è stato modificato rimuovendo la negazione nella condizione di riga 5

- A. Descrivere cosa comporta tale modifica nell'esecuzione del programma
- B. Quale o quali sono le problematiche che si creano legate alle proprietà di Mutua Esclusione, No-Deadlock e No-Starvation? Se e quali vengono violate? Motivare la risposta.

Risposta

ME:

Per accedere alla sezione critica è necessario che 1) o $k=i$ o che $k \neq i$ e il processo k sia interessato
2) solo i ha passato quando fa il controllo del ciclo for

Se a causa dell'errore più processi passano il ciclo for, vengono re-inviati al ciclo while di riga 3.

Nonostante l'errore, l'algoritmo prova a risolvere, come nell'algoritmo originale.

ME: Possono esistere sequenze di scheduling tali che un processo acceda alla CS. In tal caso altri processi che settano k dopo non possono accedere perché c'è almeno un altro passed (quello in CS)

Esempio di accesso:

interested[1]=interested[2]=true

$k=1$

entrambi $p1$ e $p2$ sono a 3

1: esce dal ciclo

2: entra nel while

2: setta passed = false

1: passed[1] = true

1: esegue 8-9 su 2 (altri eventuali processi non interessati ovviamente non sono passati)

1: esce dal ciclo for ed è pronto per la CS

2: essendo interested[1]==true

2: $k=2$

2: passed[2] = true

2: essendo passed[1]== true, va a 3

...

1: esce da CS

2: nel frattempo cicla da 3 a 9

1: mette passed[1]=false, interested[i]= false

2: riesce a entrare in CS

La ME è quindi garantita perché il secondo ciclo risolve l'errore del primo

ND:

Finché un processo solo è interessato (se $k \neq i$) non potrà accedere alla CS. Quando però il processo k setterà interested[i]= True, entrambi possono settare passed a true. Stesso ragionamento della ME, tra tante volte che tornano indietro può capitare che lo scheduler esegua una sequenza che permette a uno di accedere (Ricordatevi che un processo dopo aver eseguito la CS torna alla NCS e prima o poi uscirà da essa). Il No Deadlock è quindi garantito. La starvation invece sarà presente perché sotto determinate condizioni la sezione critica è disponibile ma i processi non riescono a procedere.