**Tehnici Programare Multiprocesor**

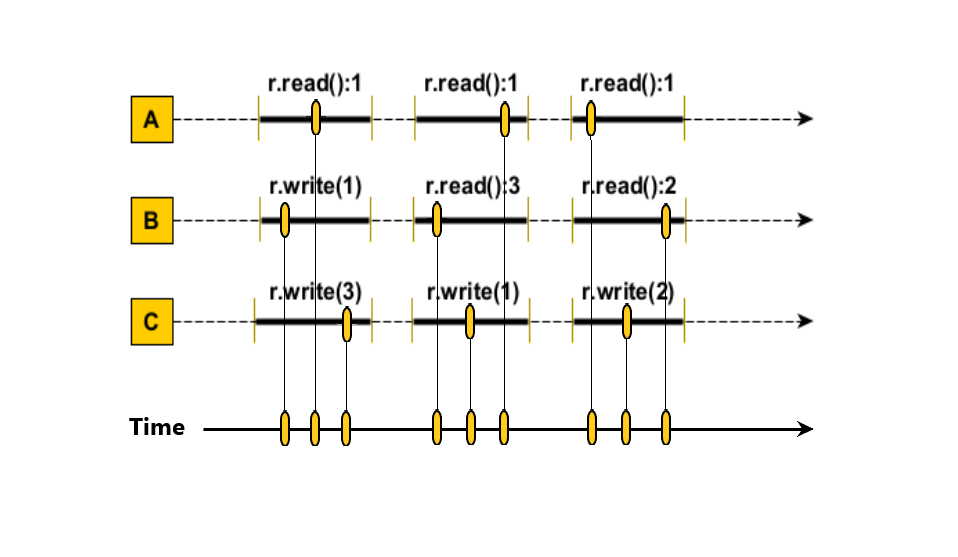
**- Tema 1 -**

Găină Robert-Adrian

Viziteu David-Andrei

Enia Vlad Ieftimie

1. Alegând punctele de liniarizare precum in diagrama de mai jos, se observă că secvența de execuție este liniarizabilă și, implicit, consistent secvențială:



* 1. Pentru a putea ilustra problema legată de algoritmul candidat pentru coada limitată folosind lock pentru n thread-uri producător și consumator, am adăugat următoarele:
     1. Coada are pe pozițiile goale valoarea -1. Astfel, ea este inițializată cu -1 peste tot, iar atunci când se face deque pe o poziție, valoarea inițială este înlocuită în queue cu -1.
     2. În metoda deq(), am adăugat un print pentru situația în care se încearcă să se scoată din coadă o valoare de pe o poziție goală, în care afișăm valorile pentru head și tail.
     3. Totodată, am adăugat un print și pentru situația în care un thread face deq() pe o coadă goală.

În metoda main(), avem o coadă cu capacitatea de 10 elemente, 3 thread-uri producător care fac enq() de 1000 de ori cu valoarea 10 și 3 thread-uri consumator care fac enq de 1000 de ori

Rezultatul execuției este următorul:

Thread [1] has locked the q for dequeue despite it being empty. head=1 tail=11

Thread [2] has locked the q for dequeue despite it being empty. head=11 tail=11

item == -1; head=12 tail=11

item == -1; head=13 tail=11

thread [3] got -1 from queue

thread [3] got -1 from queue

item == -1; head=14 tail=11

thread [3] got -1 from queue

item == -1; head=15 tail=11

thread [3] got -1 from queue

item == -1; head=16 tail=11

. . .

thread [3] got -1 from queue

item == -1; head=2073 tail=11

thread [3] got -1 from queue

item == -1; head=2074 tail=11

thread [3] got -1 from queue

item == -1; head=2075 tail=11

. . .

item == -1; head=2996 tail=11

thread [3] got -1 from queue

item == -1; head=2997 tail=11

thread [3] got -1 from queue

item == -1; head=2998 tail=11

thread [3] got -1 from queue

item == -1; head=2999 tail=11

thread [3] got -1 from queue

item == -1; head=3000 tail=11

thread [3] got -1 from queue

Se poate observa că, deși coada este goală, thread 1 si thread 2 reușesc sa facă lock pe coadă și mai apoi deq(), iar thread 3 face deq() fără oprire, valoarea head-ului crescând cu mult peste valoare tail-ului și deci peste valoarea capacității cozii.

* 1. În situația în care două thread-uri apelează metoda lock() în același timp, este posibil ca în urma apelului max(label[0], ... , label[n-1]) din cele două thread-uri să se deruleze simultan. Astfel, există posibilitatea ca rezultatul metodei max() să fie același în ambele thread-uri (așadar ambele thread-uri primesc același label), pentru că dacă totul se derulează simultan, nu a avut loc asignarea pentru label[i] în vreun thread atunci când se apelează max().

Așadar, se poate să existe două etichete egale, iar în cadrul verificării condiției din while, dacă comparăm doar etichetele, atunci cele două thread-uri cu etichete egale vor primi acces la resursa partajată în același timp.

* 1. Situația problematică apare atunci când operația lock() nu se execută cu succes și se aruncă o excepție.

Dacă acest fenomen are loc într-un block try, atunci se continuă direct în finally, unde se încearcă unlock() pe o resursă asupra căreia nu s-a executat în prealabil lock() cu succes, situația rezultată fiind una de tipul undefined behaviour.

În schimb, dacă lock() eșuează și aruncă excepție în afara unui block try, atunci execuția programului se încheie fără a mai apela funcția unlock().