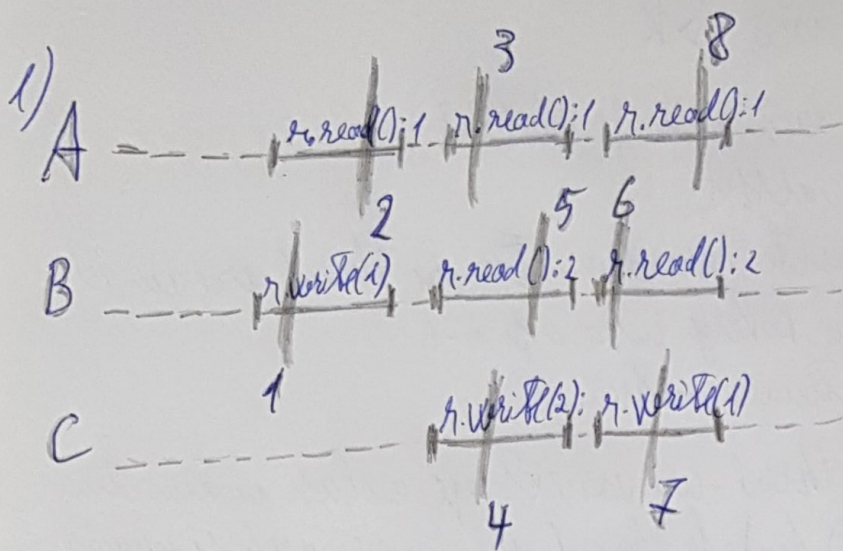


# GRUPA TPM1.

## Temă TPM



Secvența este linearizabilă

H: B r.write(1)

B r: void()

A r.read()

A r: 1

B r.read()

B r: 1

C r.write(2)

C r: read

B r.read()

B r: 2

B r.read()

B r: 2

C ~~write~~ r.write(1)

C r: void

A r.read()

A r: 1

Deoarece faptul că secvența istorică este linearizabilă, putem spune și că este consistent secvențială.



2. b) Tupla  $(label[i], i)$  reprezintă un mod de ordonare a thread-urilor

$(label[i], i) > (label[k], k) :$  True, dacă  $i > k$   
 True, dacă  $i = k$  și  $label[i] > label[k]$   
 Fals, altfel

Array-ul label poate fi citit de mai multe thread-uri. Fiecare thread are un id al lui, setat în contextul algoritmului Bakery între 0 și  $n-1$ .

Fie 2 thread-uri care vor să acceseze secțiunea critică.

Ambele apelează metoda lock. Cele 2 thread-uri vor extrage același număr din array-ul label la care vor adăuga 1 și ambele thread-uri vor avea același număr de ordine / ticket. La execuția while-ului din lock, unul dintre thread-uri îl va vedea pe celălalt că este interesat să acceseze SC, dar un are un număr de ordine sau ticket mai mic. Prin urmare va trece de while și va intra în SC. La fel și celălalt thread va acționa.

c) Apelul lock() poate arunca excepție, prin urmare obținerea lock-ului nu se mai realizează. Dacă lock() se realizează în try { și aruncă excepție, lock-ul nu se realizează și se trece imediat în blocul finally {}, unde se apelează unlock(). Cum lock() a eșuat, unlock() va eșua și el, aruncând excepție.

a) Nu funcționează corect implementarea.

Thread 1: lock

Thread 2: while (tail == head)

Thread 3: while (tail == head)



head = tail

Thread 1: insert, unlock

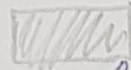
Thread 2: while (tail == head)

Thread 3: while (tail == head)

obj



head

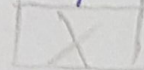


tail

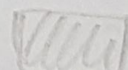
Thread 2: lock

Thread 3: lock

obj



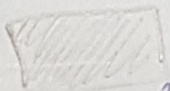
head



tail

Thread 2: extracts, unlock

Thread 3: lock



head = tail

Thread 3: extracts => Exception: No element at index head



head = tail

În mod normal nu ar trebui să extragă nici un element. Extrage o valoare a fost condus în coadă. ~~lanșarea minie~~



Considerăm cazul cu 3 thread-uri: unul enqueue și 2 deque. Thread-ul cu enqueue iese din while deoarece limita cozii nu este atinsă (0 elemente) și obține lock-ul. Celelalte 2 thread-uri sunt încă în așteptare deoarece coada încă nu conține nici un element.

Thread-ul cu enqueue inserează x, actualizează tail și eliberează lock-ul. Cele 2 thread-uri cu deque iese din acel while și vor să obțină lock-ul pe coadă. Unul dintre ele obține lock, actualizează head-ul, eliberează coada și returnează. Ultimul thread obține lock-ul, dar nu va verifica iar dacă există vreun element. Va extrage un element ca în mod normal nu s-ar afla în coada curentă.