1. Secventa de executie este si linearizabila, si consecvent secventiala. Diagrama cu punctele de linearizare si secventa istoriei de executie ar putea fi:

B: r.write(1)
A: r.read():1
A: r.read():1
C: r.write(2)
B: r.read():2
B: r.read():2
C: r.write(1)
A: r.read():1

Data aceasta istorie de executie, putem afirma ca istoria este liniarizabila. Odata ce am demonstrat ca secventa este liniarizabila inseamna ca aceasta este consecvent secventiala.

2.

a. Generalizarea propusa nu functioneaza corect. In continuare propunem un scenariu care demonstreaza un comportament nedorit pentru accesul concurent la elementele cozii. Presupunem un numar de n=3 thread-uri. Thread-urile vor executa, in ordine, urmatoarele operatii:

thread-ul 1:
t1: q.deq()
t1: q.deq()
t1: q.enq()

thread-ul 2:
t2: q.enq()

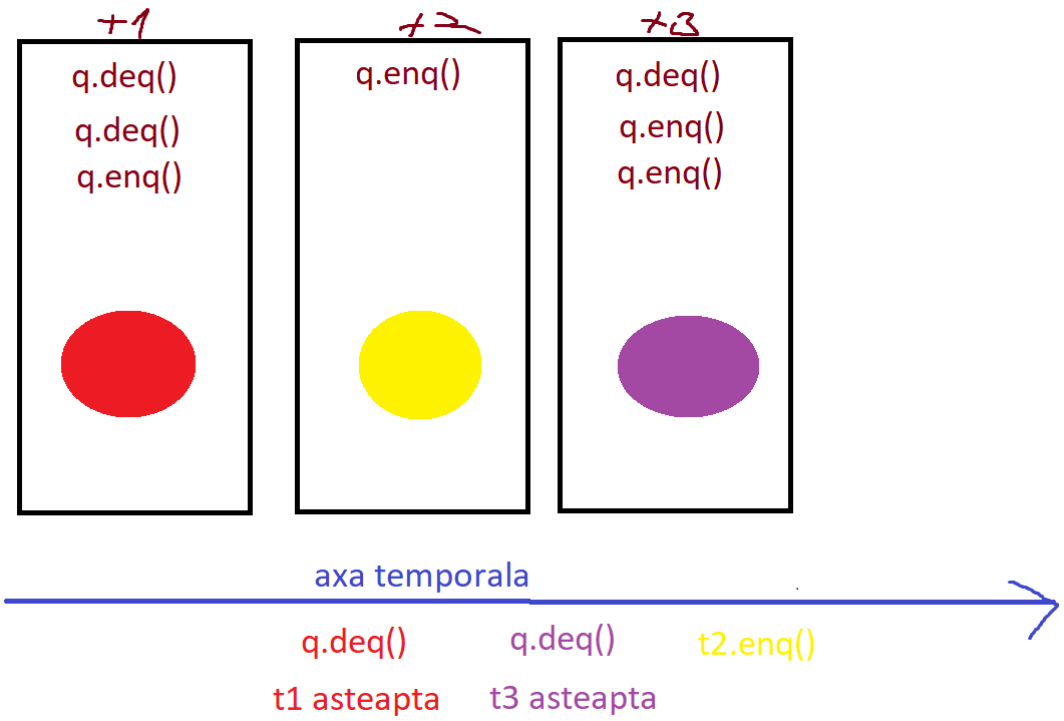
thread-ul 3:
t3: q.deq()
t3: q.enq()
t3: q.enq()

Vom considera urmatoarea ordine de executie:
t1:q.deq() → t1 va astepta in bucla while
t3:q.deq() → t3 va astepta in bucla while

t2:q.enq() → lock acquired, se adauga elementul in coada, lock released

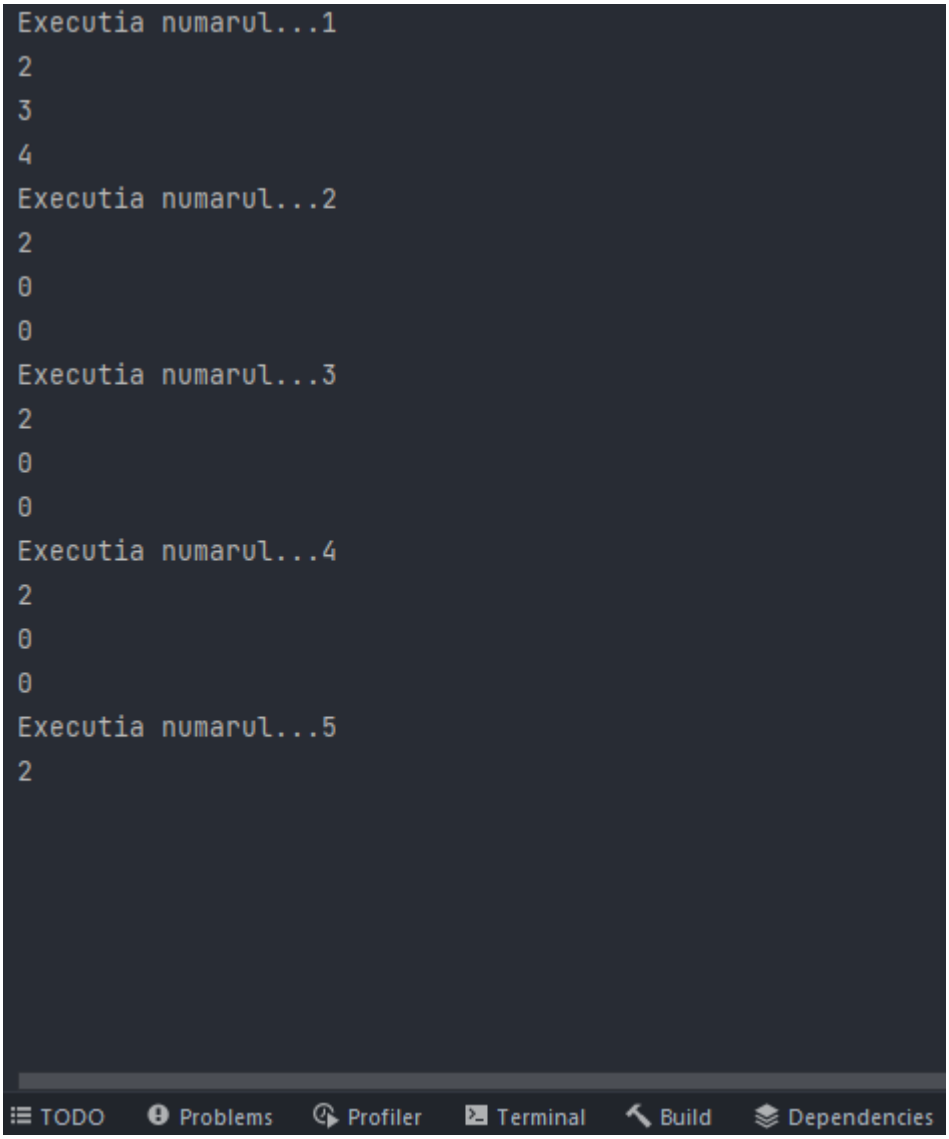
Dupa terminarea apelului enq din thread-ul t2, thread-urile care asteptau (t1 si t3) datorita faptului ca tail == head (coada era vida si nu se putea executa operatia de deq) vor sesiza schimbarea la nivelul capetelor cozii si astfel vor iesi din bucla while care le obliga sa astepte.

In pasul urmator, thread-urile vor incerca sa acceseze lock-ul in mod exclusiv. Presupunem ca urmatorul thread care va obtine lacatul este t3. T3 va executa deq iar coada va ramane vida. Din nou, thread-urile vor incerca sa acceseze in mod exclusiv resursa partajata si presupunem ca urmatorul thread va fi t1. In acest moment tail == head (== 1) ceea ce inseamna ca queue este goala. Dar deja t1 a facut lock pe resursa si executa deq ce va duce la coruperea cozii si a firului logic al executiei intrucat head > tail iar elementul ce va fi returnat din apel este unul invalid.



Exemplul prezentat in programul rulat este acelasi de mai sus insa pentru enq am folosit:

T1.enq(1)
T2.enq(2)
T3.enq(3)
T3.neq(4)


Executia numarul...1
2
3
4
Executia numarul...2
2
0
0
Executia numarul...3
2
0
0
Executia numarul...4
2
0
0
Executia numarul...5
2
TODO
Problems
Profiler
Terminal
Build
Dependencies

b. Presupunem algoritmul Bakery fara compararea id-ului thread-ului (compararea exclusiva a label-ului). Consideram 2 thread-uri, t1 si t2, care incep executia functiei lock() in acelasi timp. Ambele thread-uri isi seteaza flag-ul corespunzator id-ului ca fiind true, apoi vor calcula label-ul asociat care va deveni 1 (intrucat nu a mai fost intalnit niciun thread pana la momentul respectiv). Se va intra in bucla while si spre exemplu, thread-ul 1 va gasi k=2 cu flag-ul setat pe true iar label[k] va fi egal cu label[i], deci se va iesi din while si se va accesa sectiunea critica. De asemenea, thread-ul 2 la randul lui intra in bucla while si gaseste k = 1 cu flag[i] = true iar label[k] = label[i] si acesta va iesi din while si va accesa seciunea critica. Astfel, in acest caz este incalcat accesul exclusiv la sectiunea critica.

Daca luam in considerare si id-urile thread-urilor, intrucat ele sunt unice va exista o anumita ordine (tipul este int, si exista ordinea crescatoare pe numerele intregi) si astfel accesul ii va fi oferit doar unui singur thread.

c. In exemplul 1, daca apelul functiei lock() va produce o exceptie (din diferite motive) nu vom fi constransi sa facem unlock asa cum se intampla in exemplul 2. In exemplul 2, daca apelul functiei lock va arunca o exceptie atunci imediat dupa terminarea block-ului try se va executa finally si anume se va face unlock() pe lock-ul care a dat o exceptie.

Apelul functiei unlock pe lock-ul care a aruncat o exceptie poate deregla comportamentul lacatului, aducand inconsistente la accesul exclusiv al sectiunii critice.

3.

a) La subpunctul a) am creat mai multe threaduri care doresc sa manance din oala si un thread bucatar care va face reumpleri de un numar limititat de ori pentru a fi siguri ca acest proces se va termina.(numarul de ori este calculat in functie de cati salbatici care vor sa manance avem si de numarul de portii pe care acestia doresc sa il serveasca). Toate threadurile vor dori acces exclusiv la oala, bucatarul sa stie daca aceasta este goala si sa o reumple iar salbaticii pentru a manca. Astfel, toti vor concura pentru un lock care va oferi acces exclusiv. In cazul in care un salbatic doreste sa manance iar oala nu este goala, atunci acesta se va servi din oala si isi va termina executia. Altfel, daca acesta vrea sa se serveasca din oala iar aceasta este goala, salbaticul respectiv va mai astepta si incerca din nou sa manance peste putin timp. Bucatarul pe de alta parte, va concura si el pentru accesul exclusiv la oala. Daca aceasta nu este goala, nu va schimba state-ul oalei insa daca aceasta va fi goala o va reumple cat timp vor mai fi mancatori care doresc sa se serveasca din oala.
b) Pentru a avea o executie cat mai fair vom pune fiecare salbatic care vrea sa se serveasca din oala „la rand”. Prin acest lucru ma refer la faptul ca mereu cand vom avea mai multi mancatori care vor sa acceseze oala, il vom lasa pe cel care a asteptat cel mai mult. Astfel, vom garanta ca niciun salbatic nu se va pune in fata celorlalti si va manca cat doreste pana isi satisface numarul de portii. Acest lucru este dat de faptul ca mereu dupa ce un salbatic mananca, acesta trebuie sa se pune din nou la rand pentru a accesa din nou oala.

Comparatie intre cele doua abordari pentru numarul de portii mancate de un salbatic = 1

#threaduri_mancaciosi	#reumple_bucatar	Timp var unfair	Timp var fair
4	1	3	4
4	1	3	5
4	1	6	5
4	1	3	4
4	1	2	5
4	2	2	3
4	2	6	4
4	2	4	4
4	2	2	5

4	2	3	4
8	1	4	4
8	1	6	4
8	1	6	5
8	1	4	7
8	1	4	6
8	2	2	4
8	2	5	4
8	2	4	4
8	2	3	7
8	2	4	3
16	1	15	9
16	1	9	14
16	1	8	7
16	1	14	13
16	1	7	12
16	2	6	9
16	2	5	8
16	2	6	8
16	2	6	8
16	2	18	7

Exemple de rulari pt b)

#portii per salbatic	#threaduri_salbatici	#reumple_bucatar	Timp var fair
100	16	4	70
100	16	4	73
100	16	4	71
1000	16	4	559
1000	16	4	616
1000	16	4	571
10000	16	4	5218
10000	16	4	5092
10000	16	4	5057

#portii per salbatic	#threaduri_salbatici	#reumple_bucatar	Timp var fair
100	8	4	32
100	8	4	29
100	8	4	30
1000	8	4	168
1000	8	4	160
1000	8	4	180
10000	8	4	1294
10000	8	4	1453
10000	8	4	1391