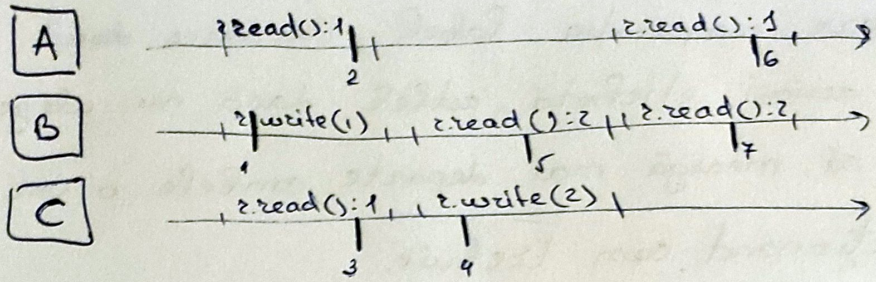
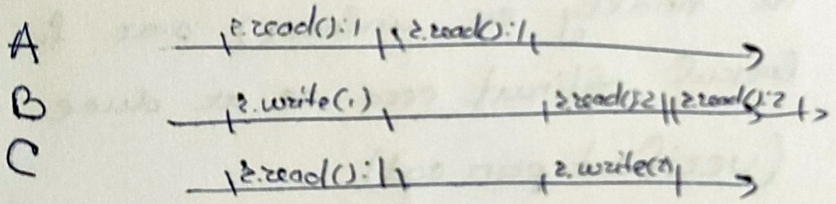
1.



Pe diagramă am pus punctele de linearizare și putem deduce că secvența nu este linearizabilă întrucât citirea de la punctul de linearizare 6 are loc după citirea de la punctul 5 și observăm că rezultatele citirilor diferă. Însă pentru linearizabilitate știm că dacă un read returnează ceva, toate readurile următoare trebuie să returneze la fel (dacă nu există vreun write, ca și în cazul nostru).

Totodată secvența este consistent secvențială:



Am păstrat aceiași ordine de execuție în cadrul A,B,C și întrucât ordinea între procesoare este nedefinită, al doilea read():1 de la A poate fi executat și înainte la z.write(2) (C), astfel obținând consistența secvențială.

b)
Nu este suficientă doar comparația label, deoarece două threaduri pot obține aceiași etichetă, astfel dacă nu alegem care din ele două să meargă mai departe, ambele o vor face. Lock-ul nefuncționând cum trebuie.

Ex:

thread 1 : flag[1] = true, label[1] = 3
thread 2 : flag[2] = true, label[2] = 3

prima parte a while-ului e ok, însă la label[i] > label[k] avem contradicție 3 > 3, thread 2 - neblocat. astfel ambele
Fals threaduri vor accesa secțiunea critică.
de aia e nevoie de departajarea prin thred_id/nr de ordine.

c) Apelul lock() înainte de blockul try căci apelul lui lock() poate arunca excepții astfel nu se va executa, însă dacă ar fi fost în try, blocul de finally cu unlock() s-ar fi executat chiar și fără lockul obținut. ceea ce ar duce la eroare în program. (verificat prin cod).

a) Pentru DoubleLockBasedQueue generalizarea funcționează bine căci fiecare thread are acces mutual exclusiv la secțiunea critică asigurate de cele două lockuri. Deoarece enqueue() manipulează doar tail și dequeue() manipulează doar head nu e necesar să fie folosit același lock. Totodată, fiecare thread așteaptă în cazul în care coada e plină (pt. enqueue()) să se elibereze. dar și în cazul în care este goală (pt. dequeue()) să se adauge ceva în ea.