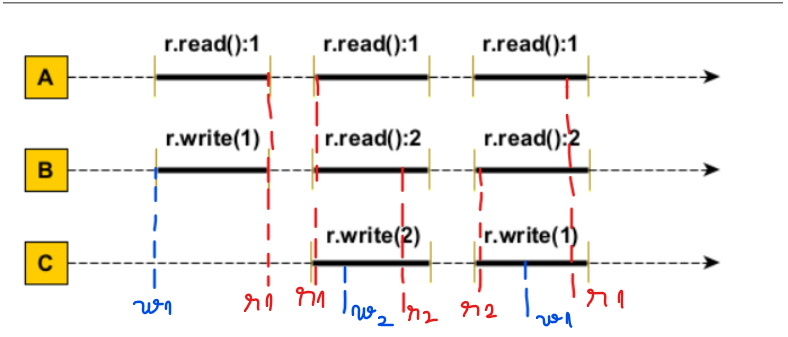
Tema

Tehnici de programare multiprocesor

Roman Ștefan & Anicăi Denis - George

# Ex. 1

Secvența dată este linearizabila deoarece fiecare operație de read() se realizeaza după operație de write() valoarea corespunzătoare (w1, r1, r1, w2, r2, r2, w1, r1) .

Deoarece este linearizabila, secvența dată este și consistent secvențială.

# Ex. 2

Apelul metodei lock() se apelează înaintea blocului try deoarece :

* Nu ar trebui să folosim datele dacă nu putem obține lock-ul;
* Dacă apelăm lock() în interiorul blocului try iar metoda nu reușește, se poate întâmpla să aruncăm o excepție în blocul try, iar apoi una in block-ul finally deoarece apelăm unlock fără ca lock-ul sa fie obținut.

# Ex. 3

## a)

Shady lock-ul poate duce la un deadlock deoarece toate thread-urile pot ramane blocate, astepandu-se unele pe celelalte deoarece atunci cand un thread seteaza variabila turn pe me, si used pe true iar apoi incearca sa iasa din do while(turn != me), un alt thread poate schimba valoarea variabilei turn iar iesirea din do while nu va mai avea loc.

## b)

Very Shady Lock-ul nu asigura garantie de fairness, deoarece sa presupunem o execuție cu 4 threaduri : 1, 2, 3, 4. Threadul 1 va seta x = 1, y = 1. Intre timp threadurile 2,3,4 vor schimba variabla x si vor aștepta la pasul while (y!=0){}. Dupa ce threadul 1 va apela unlock(), oricare dintre threadurile 2, 3 ,4 poate să iasă din while și sa schimbe variabila y, fara a avea o ordine. Intre timp, threadul 1 se poate sa ajungă in bucla while(y!=0){}, iar cand metoda unlock() va fi apelată el poate sa iasa din nou înaintea celorlalte, astfel va avea mai multe accesari la secțiunea critică.

## c)

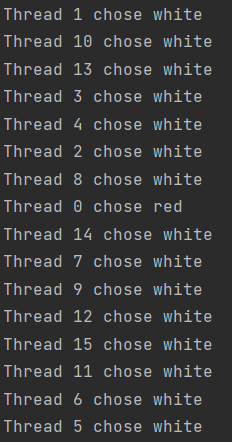
Cazurile care trebuie verificate sunt :

i) Daca un thread va seta getWhite = true si va ajunge la if(last == me) înainte ca celelalte threaduri sa modifice valoarea last, atunci acel thread va avea valoarea “red” in timp ce toate celelalte vor avea valoarea “white”;

ii) Dacă celelalte threaduri vor schimba valoarea last înainte ca primul thread să ajungă la if(last == me) atunci acela va avea valoarea “black”, în timp ce toate celelalte vor avea valoarea white, deci nimeni nu va avea valoarea “red”;

iii) Dacă mai multe threaduri trec de if(getWhite) înainte ca unul din ele sa modifice getWhite = true(sa presupunem ca 3 threaduri au trecut de if(getWhite = true, atunci primele vor avea valoarea “black” deoarece al 3-lea thread a schimbat valoarea last, iar al 3-lea thread poate avea fie “red” fie “black”, dacă a mai schimbat cineva variabila last între timp), iar restul vor avea valoarea “white”

Deci, maxim un thread poate avea valoarea “red”, iar maxim n-1 pot avea valoarea “white”.



În majoritatea execuțiilor, cazul i) este intalnit.

# Ex. 4

Link-ul problemei : <https://github.com/StefanRoman1/Tema---TPM/tree/main/src>

# Ex. 5

Algoritmul generalizat al lui Petterson nu este fair deoarece nu garantează că firele de execuție vor fi executate în ordinea în care au fost pornite. Acest lucru se datorează faptului că firele nu se așteaptă unul pe celălalt să se termine, ci ca victima să se schimbe. Deci, dacă începe un thread cu un nivel mai înalt, nu va aștepta ca celelalte fire să se termine, ci ca victima să se schimbe. Aceasta înseamnă că firele de execuție cu un nivel inferior nu vor fi executate în ordinea în care au fost pornite. Un fir poate fi executat de mai multe ori înainte ca un alt fir să fie executat chiar și o singură dată.