Tema 5 – PPD

Timpi de executie

Caz de testare	Timp
P_r=4, P_w=2	84.75ms
P_r=4, P_w=4	61.55ms
P_r=4, P_w=12	62.26ms

Concluzii

Cresterea numarului de workeri este favorabila pana la un anumit punct, dupa care se observa efectele overheadului, insa fara a afecta prea mult performanta.

Explicatii

Lista inlantuita

Fiecare nod are un mutex propriu. Am folosit lista inlantuita cu santinele pentru simplificarea implementarii. Inserarea unui nod in lista pastreaza ordinea elementelor. Operatia de update se descompune in extragerea (stergerea) nodului din lista, incrementarea punctajului si reinserarea acestuia.

- Inserarea unui nod N
    - Blocam n0=start, n1=start.next (poate fi end sau primul nod din lista)
    - Daca n1==end, se insereaza N intre n0 si n1 si am terminat
    - Daca n1 este un element in lista, verificam daca N<n1 si, in caz afirmativ, inseram N intre n0 si n1
    - In caz contrar, iteram prin blocari successive (lock n1.next, unlock n0, n0=n1, n1=n1.next) pana cand n0<N<n1, caz in care il inseram pe N intre n0 si n1
    - Se acorda atentie deblocarii nodurilor ramase blocate la finalul metodei
- Cautarea unui nod N dupa o properietate (ex. id)
    - Se foloseste ideea cu blocari successive pentru nodul current si nodul urmator pana cand gasim un nod care indeplineste conditia
- Stergerea unui nod dupa o proprietate
    - Se modifica mecanismul de cautare folosing iterarea prin blocarea a 3 noduri succesive la un moment dat. Verificam initial daca lista este vida (start->end), caz in care nu facem nimic.
    - Blocam n0=start, n1=start.next, n2=n1.next. Consideram n1=nodul current (folosit in parcurgere) si n0,n2 vecinii sai.
    - O parcurgere totala a listei se termina cand n1=end ⇔ n2=NULL.
    - Trecerea la urmatorul element se efectueaza astfel:
        - unlock(n0)
        - n0=n1
        - n1=n2
        - n2=locked(n2.next)
    - Atunci cand n1 satisface conditia sata, este eliminate din lista, deblocat, iar legatura n0->n2 este ajustata si se incheie metoda

- Gestionarea participantilor eliminate
    - Se foloseste un set<int> pentru a retine id-urile “blacklisted”. Orice operatie pe aceasta structura (insert/find) este protejata de un lock.
- Gestionarea situatiei in care 2 threaduri insereaza simultan acelasi id
    - Pentru a evita riscul duplicarii, se foloseste un map<ID, mutex>. La orice operatie pe un anumit id, se blocheaza mutexul corespunzator, astfel incat threadurile care actioneaza asupra aceluiasi id sa se sincronizeze. Threadurile care actioneaza asupra unor id-uri diferite ruleaza in regim normal, fiind influentate doar de lock-urile proprii nodurilor

Coada

- Se folosesc doua variabile conditionale care au in spate acelasi mutex:
    - cv_empty pentru a bloca coada vida la dequeue
    - cv_full pentru a bloca coada plina (capacitate maxima) la enque
- Datorita restrictiei de capacitate maxima, s-a ales folosirea unui container linear (array) pentru a implementa coada
- enqueue(item)
    - lock(mutex)
    - cat timp (coada este vida) cv_full.wait()
    - <inseram item in coada>
    - cv_empty.notify_one()
- dequeue(&item):bool
    - lock(mutex)
    - cat timp (coada este goala **si activa**) cv_empty.wait()
    - daca (coada nu mai este active si este goala) return FALSE
    - item = <extrage item din coada>
    - cv_full.notify_one()
    - return TRUE
- Pentru a modifica proprietatea de **coada activa** se foloseste un mecanism de shutdown care precizeaza ca nu se vor mai insera elemente noi in coada, dar va fi posibile extragerea elementelor deja existente.