**Analiza**

* Spre deosebire de implementarea laboratorului anterior, unde blocările se realizau pe întreaga listă, acum, nivelul de granularitate devenind unul fin, la nivel de nod, performanța programului multithreading este cu mult îmbunătățită (nemaiavând in place mecanismul busy-waiting, care, de asemenea, consuma resurse CPU).

|  |  |  |
| --- | --- | --- |
| **Nr readers** | **Nr workers** | **Timp executie total** |
| Secvential | cu timpul | 344.3 |
| 4 | 2 | 40.02 |
| 4 | 4 | 26.98 |
| 4 | 12 | 19.16 |

**Implementare**

* Secvențial
  + Definesc o structură de date care să simuleze o listă dublu înlănțuită, având ca dată un Participant (struct cu 3 proprietăți: id, punctaj și țară). La inserarea unui element în listă, se verifică hash-ul participantului. Dacă acesta a fost salvat într-un vector de hash-uri, atunci nu se mai face nimic (înseamnă că a fost eliminat din concurs acel concurent). Altfel, dacă a mai fost adăugat, i se incrementează scorul și se resortează lista, ori se adaugă pentru prima dată și se sortează lista. De fiecare dată când vom încerca să introducem un participant cu punctajul -1, acesta va fi marcat prin adăugarea hash-ului în vectorul amintit anterior și eliminarea (dacă acesta deja exista în listă) lui din listă.
* Paralel
  + Pe lângă structurile descrise mai sus, se adaugă încă una pentru implementarea unei cozi cu excluderi mutuale, atât la push (insert), cât și la pop (get), respectiv la interogarea coadă vidă/ final citiri din fișiere. Totodată, această coadă are două variabile condiționale, atât pentru insert, cât și pentru get, astfel că se poate face push doar dacă în coadă nu sunt mai mult de 50 de elemente (o constantă), iar pop se poate face dacă nu este coada goală sau mai sunt threaduri care produc (cititorii). În momentul în care un cititor termină, sunt notificați toți consumatorii.
  + Aici am implementat nu o listă dublu înlănțuită, ci una simplu înlănțuită. Menționez că folosesc o santinelă pe post de cap, cu id = -1. Strategia este următoarea: Păstrăm logica unui vector de hash-uri, pe care punem un mutex când inserăm o nouă valoare, foarte important, pe care îl eliberăm abia după ce un nou mutex este blocat pe resursa nodului cap al listei. Cât timp nu am ajuns la finalul listei și cât timp nu am găsit vreun nod cu id-ul participantului de adăugat/actualizat/șters, obțin un lock pe nodul curent (inițial = următorul nod de după cap), dacă are id-ul = id-ul participantului, știu că fusese deja inserat în listă. În cazul în care are un scor pozitiv, incrementez valoarea scorului din listă, altfel șterg nodul. La finalul lui while, eliberez blocarea pe nodul anterior, acesta ia valaorea nodului curent, iar nodul curent ia valoarea următorului nod. La ieșirea din while, dacă constat că nu există un nod cu id-ul participantului și că acesta are un scor pozitiv, atunci îl adaug la finalul listei. Eliberez mutexul de pe nodul anterior (care, în acest punct, este ori ultimul nod din listă, ori unul de eliminat. Iar dacă am găsit un nod în listă cu id-ul participantului și acesta are un scor negativ, atunci acel nod anterior este cel pe care l-am șters virtual, refăcând conexiunile, iar acum pot elibera memoria.
  + La final, sortez lista și o afișez, fără acel nod santinelă.