**Documentatie – Laborator 5**

Run:

* Se utilizeaza un executor de tip FixedThreadPool pentru indeplinirea task-urilor de citire din fisier. Un task pentru fiecare fisier.
* Fiecare task de citire citeste cate un entry din fisier si il pune in coada. Metoda de adaugare este sincronizata prin folosirea unui lock la apelarea metodei. In interior se verifica daca acea coada a atins capacitatea maxima, caz in care se asteapta pe variabila conditionala *notFull*. In caz ca nu este atinsa capacitatea maxima, se adauga entry-ul in coada dupa care se notifica trezirea thread-urilor worker care asteapta sa elimine un element din coada pentru a il procesa prin utilizarea variabilei conditionale *notEmpty*.
* Pentru eleminarea unui entry din coada se verifica prima data daca este goala, caz in care se asteapta pe variabila conditionala *notEmpty*. Daca exista un entry, se eilimna, se notifica eventualele threaduri care doresc sa puna in coada elemente ca aceasta nu mai este plina prin folosirea variabilei conditionale *notFull*, dupa care incepe procesarea elementului.
* Pentru procesare, prima data se verifica daca id-ul entry-ului nu se afla in lista celor eliminate (se foloseste un concurrent hash map). Daca scorul este -1 atunci se adauga id-ul in lista respectiva dupa care se incepe parcurgerea listei cu clasamentul. Pentru parcurgere se de lock mereu la nodul anterior si la cel curent mereu in ordinea aceasta.
* Daca nodul curent are acelasi id atunci:
  + La update se actualizeaza scorul dupa care se elibereaza lockul pe noduri
  + In cazul de delete se incearca prima data blocarea nodului urmator dupa cel curent, dupa care se face stergerea propriu-zisa si se elibereaza lockurile.
* Daca nodul nu are acelasi id se continua cautarea prin eliberarea lockului pe nodul anterior, mutarea nodului curent in cel anterior si blocarea urmatorului nod.
* Pentru simplificarea parcurgerii s-a folosit o lista inlantuita cu santinele.
* Cand toti readeri si-au terminat executia se notifica threadurile worker pentru a nu ramane blocate in asteptarea unor elemente noi.
* Daca readeri au termin de citit toate fisierele si queue-ul este gol atunci workeri isi incheie si acestia executia.
* Cand toate thread-urile si-au terminat executia, se sorteaza lista dupa care se scriu elementele acesteia in fisier.

**Testare:**

**Testare Java:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nr tari | Nr probleme | Queue Size | Nr threads | Reader thrads | Timp executie |
| 5 | 10 | 50 | 6 | 4 | 93.4441 |
| 8 | 100.1037 |
| 16 | 102.4960 |
| 100 | 6 | 97.2675 |
| 8 | 103.4915 |
| 16 | 101.5080 |

**Analiza:**

*Testarile s-au realizat pe un CPU Intel I3 12100F 4 cores 8 threads.*

* **Java**

Se poate observa ca timpii obtinuti sunt mult mai mari comparativ cu cei de la laboratorul trecut. Aceasta se poate datora in primul rand noilor conditii impuse problemei cum ar fi capacitatea maxima a cozii. Citirea fiind mult mai rapida decat procesarea unei intrari, se ajunge foarte des la situatia in care coada este plina iar threadurile care se ocupa de citire sunt nevoite sa astepte eliberarea cozii pentru a putea continua citirea. De asemenea, desi teoretic utilizarea unui mecanism de fine grain synchronization cu un lock pe fiecare nod ar duce la o imbunatarire a gestionarii concurentei prin permiterea mai multor threaduri sa parcurga simultan lista, aceasta imbunatatire nu se poate observa, ba chiar se intampla chiar opusul, deoarece vor fi situatii in care pentru parcurgere mai multe threaduri vor dori sa blocheze acelasi nod pentru a verifica informatia din acesta si astfel vor trebui sa astepte unul dupa celalalt, desi poate nu trebuie sa indeplineasca nicio actiune pe nodul respectiv. Mai mult, mecanismul de lock este unul costisitor si duce la cresterea timpului de executie.

