**Documentatie – Laborator 4**

Run option:

0 – secvential:

* Se parcurge fisier cu fisier. Se citeste fiecare intrare pe rand si se adauga in lista.

1 – rulare paralela:

* Se lanseaza thread-urile pentru citire in executie. Se face o impartire echitabila a fisierelor care trebuiesc citite intre thread-uri similar cu impartirea elementelor unei matrici din orele anterioare intr-un mod cat mai egal intre thread-uri.
* Fiecare thread reader citeste cate un entry din fisier si il pune in coada. Metoda de adaugare este sincronizata astfel incat doua thread-uri nu o pot apel simultan. In interior se adauga entry-ul in lista dupa care se notifica trezirea unui thread worker care asteapta sa elimine un element din coada.
* Dupa eleminarea unui entry din coada, acesta este procesat. 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 blocheaza lista cu clasamentul si se verifica existenta unui nod cu acelasi id prin parcurgere. Se urmeaza pasii specificati in enuntul problemei.
* Pentru mentinerea listei sortate, dupa adaugarea scorului se cauta doar in partea stanga a nodului curent nodul dupa care ar trebui sa se insereze, dupa care se sterge nodul vechi si se refac legaturile.
* 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 toti workeri si-au terminat executia se notifica primul reader care era in asteaptare sa afiseze lista finala.

**Testare:**

**Testare Java:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nr tari | Nr probleme | Nr reader threads | Nr total threads | Timp executie |
| 5 | 10 | secvential | | 64.96209 |
| 1 | 4 | 70.50592 |
| 6 | 70.52256 |
| 8 | 69.94437 |
| 16 | 70.0611 |
| 2 | 4 | 63.14347 |
| 6 | 63.76174 |
| 8 | 65.33901 |
| 16 | 63.82687 |

**Analiza:**

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

* **Java** – secvential vs paralel

Se poate observa ca variantele paralele cu 3, 5, 7 sau 15 thread-uri nu par sa ofere o imbunatatire semnificativa fata de varianta secventiala. In unele cazuri, chiar arata un timp de executie putin mai mare. Aceasta se poate datora in primul rand gradului de concurenta: daca operatiile de citire si scriere in coada sunt suficient de rapide, adaugarea de thread-uri suplimentare pentru a le gestiona poate duce la o competitie mai mare pentru resurse, ceea ce poate afecta performanta. De asemenea, in cazul in care se utilizeaza un singur lock pentru intreaga lista, operatiile pe nodurile din lista ajung sa se execute aproape secvential, anuland astfel beneficiile paralelizarii. De exemplu, daca un thread blocheaza intreaga lista pentru adaugarea unui nod, toate celelalte thread-uri vor fi blocate, indiferent daca incearca sa efectueze operatii pe noduri diferite.

