|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mode** | **Threads** | **T\_R** | **Time** | **Ratio** |
| **Sequential** | **1** | **1** | **146401005** | **1.00** |
| **Parallel** | **4** | **1** | **131078880** | **1.12** |
| **6** | **245889626** | **0.60** |
| **8** | **383391744** | **0.38** |
| **16** | **742360989** | **0.20** |
| **4** | **2** | **59257453** | **2.47** |
| **6** | **60984567** | **2.40** |
| **8** | **62634424** | **2.34** |
| **16** | **72016937** | **2.03** |

C++

Se impart fisierele in mod egal producatorilor, care citesc linie cu linie si pun rezultatele intr-o coada sincronizata, asemanator Blocking Queue din Java, sincronizarea fiind realizata manual folosind std::mutex si std::semaphore. Cand producatorii termina toata munca, pun in coada un Poison Pill, care arata ca consumatorii trebuie sa se opreasca cand il intalnesc. In rest, consumatorii iau cate un rezultate din coada si daca scorul este -1, il pun pe lista de persoane discalificate si sterg din lista rezultat, altfel daca persoana nu este descalificata se adauga/updateaza lista cu noul scor. Sincronizarea este facuta pe sectiunea critica cu ambele liste deodata, sincronizarea fiecare liste individual nu este suficienta pentru a nu garanta race condition. Lista este mereu pastrata ordonata, iterand mereu prin lista si inserand in locul pentru a mentine invariantul ordinii.

Folosirea mai multor consumatori duce la un timp mai mare de executie, din cauza overheadului lock/unlock la mutex la fiecare operatie. Folosirea mai multor producatori produce rezultate mai bune, eliminand din overheadul de citire. Per total, variantele cu 2 threaduri de citire ruleaza mai repede decat varianta secventiala, dar varianta cu un singur thread de citire functioneaza mai incet in general.