# C++

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sequential** | | | **109264446** |  |
| **p\_w** | **p\_r** | **Max Capacity** | **Time** | **Ratio** |
| **12** | **2** | **50** | **97926116** | **1.115785** |
| **100** | **102287464** | **1.06821** |
| **4** | **50** | **87528420** | **1.248331** |
| **100** | **92299210** | **1.183807** |
| **2** | **50** | **54307485** | **2.011959** |
| **100** | **65416548** | **1.670288** |
| **12** | **4** | **50** | **79505682** | **1.374297** |
| **100** | **85811115** | **1.273313** |
| **4** | **50** | **89514388** | **1.220636** |
| **100** | **82567182** | **1.32334** |
| **2** | **50** | **96495749** | **1.132324** |
| **100** | **67270204** | **1.624262** |

Ca la laboratorul anterior, 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 de data asta std::mutex si std::condition\_variable. 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 descalificate si sterg din lista rezultat, altfel daca persoana nu este descalificata se adauga/updateaza lista cu noul scor. Lista este apoi sortata la final si rezultatele afisate.

Sincronizarea este facuta la nivel de lista folosind fine grained synchronization, avand cate un mutex in fiecare nod. Cand parcurgem lista, mereu blocam nodul current pentru a ne asigura ca nu s-a sters/inserat un nod dupa nodul curent si ca avem valoarea actuala a nodului. Cand vrem sa stergem un nod, blocam si nodul inaintea caruia stergem (si cand inseram, de fapt doar aia blocam), pentru a ne asigura ca facem legatura bine. Avem sincronizarea la nivelul celor doua liste, dar acest lucru nu este suficient pentru a evita un race condition, intrucat avem instructiuni suprapuse cu operatii pe cele doua liste. Problema apare cand dam de un rezultat -1 si unul valid pentru acelasi id, si solutia este sa avem un mutex la nivel de id pe care il blocam cand dam de un anumit id. Daca tinem indexii intr-un map pentru fiecare id, avem nevoie de un alt mutex cand acesam map-ul, deoarece initial cand accesam un id pentru prima oara, se va crea un mutex nou.

Folosirea mai multor threaduri de citire aduce in general o imbunatatire in performanta, intrucat evitam fenomenul de starvation. Pentru p\_r=2, un p\_w mai mare tine sa scada din performanta (inducand starvation), pe cand pentru p\_r=4 un p\_w mai mare tind sa creasca performanta (eliminand din starvation). O capacitate mai mare a cozii a fost cu foarte putin mai inceata pentru p\_r=2 si cu foarte putin mai rapida pentru p\_r=4. Cea mai rapida varianta a fost p\_r=p\_w=2, datele fiind mici si overheadul sincronizarii fiind mare comparativ cu procesarea efectiva.