Laborator 4

Obiectiv:

* Intelegerea/aprofundarea sablonului “producator-consumator”
* Intelegerea/aprofundarea sincronizarii conditionale
* Intelegerea/aprofundarea excluderii mutuale (granularitatea sectiunilor critice)

La un concurs de programare participa mai multi concurenti din diferite tari. Concursul presupune rezolvarea mai multor(10) probleme intr-un interval de timp dat. Fiecare participant este identificat printr-un ID si rezultatele obtinute pentru rezolvarea fiecarei probleme sunt salvate intr-un fisier cu rezultate in care pe fiecare linie sunt inregistrari de forma (ID, Punctaj). Punctajele concurentilor dintr-o tara C1 corespunzatoare fiecarei probleme se gasesc in fisiere separate (“RezultateC1\_P1.txt”, “RezultateC1\_P2.txt, …“RezultateC1\_P10.txt). Un punctaj negativ (-1) inseamna incercare de fraudare si va conduce la eliminarea concurentului din concurs. Daca un concurent nu rezolva o problema NU se adauga in fisier o pereche cu ID –ul concurentului si punctaj 0.

Pentru a se ajunge la clasamentul final se cere sa se formeze o lista inlantuita ordonata descrescator (dupa punctaj) care contine noduri cu valori de tip pereche (ID, Punctaj). Lista este o lista ordonata (*invariant: dupa orice operatie cu lista aceasta este ordonata – inserati in lista ordonata nu sortati dupa fiecare operație*).

Se porneste prin crearea unei liste inlantuita vida si se adauga noduri sau punctaje pe masura ce se citesc inregistrari din fisiere.

La citirea unei noi perechi (ID\_n, Punctaj\_n) se verifica daca exista deja in lista o pereche cu ID egal cu ID\_n:

* Daca exista atunci
  + Daca Punctaj\_n este pozitiv se aduna punctajul Punctaj\_n la punctajul existent in nod,
  + Daca Punctaj\_n = -1 atunci se sterge nodul gasit din lista si se adauga intr-o lista cu ID-uri de persona non-grata
    - Daca se va citi ulterior o pereche cu acelasi ID nu se va mai adauga in lista cu punctaje
* Daca nu exista atunci
  + se adauga un nou nod cu valoarea (ID\_n, punctaj\_n).

In final aceasta lista va conține clasamentul final si se va salva intr-un fisier „Clasament.txt” de catre main-thread.

Metoda A) Implementare secventiala

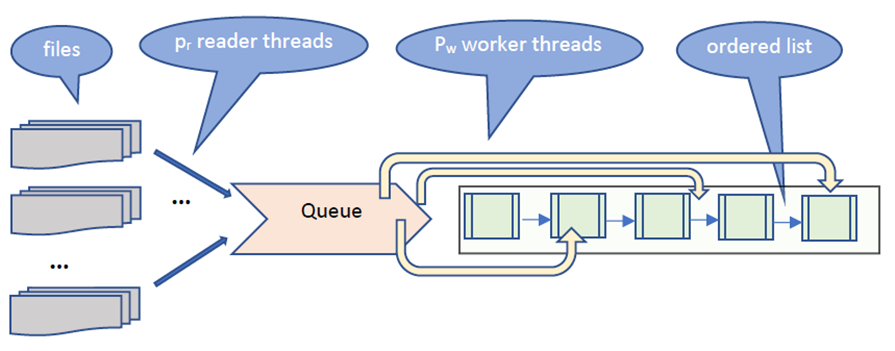
* Se citeste pe rand din fiecare fisier cate o pereche si se adauga in lista rezultat -L. La final se scriu elementele listei in fisierul rezultat.

Metoda B) Implementare paralela – p threaduri (nu se folosesc Executori!!!)

1. **p\_r** threaduri (readers) citesc din fisiere perechi (ID , Punctaj) si le adauga intr-o structura de date de tip coada care nu este marginita (nu are capacitate maxima). Politica de partajare a fisierelor intre readers trebuie sa asigura o incarcare de calcul balansata.

(conditie – pentru structura de tip coada NU se admite folosirea unei structuri de date pentru care partea de sincronizare este deja implementata!!!)

1. Celelalte threaduri p\_w= (p-p\_r) threaduri (workers) preiau cate o pereche din coada si o adauga in lista L.
   * Se continua operatiile 1., 2. pana cand toate perechile, din toate fisierele, sunt adaugate la lista L.
2. Dupa ce toate rezultatele au fost adaugate in lista L, primul thread (primul reader) scrie rezultatul obtinut in lista L in fisierul „Clasament.txt”.



**Atentie: O stare in care coada(queue) e vida nu inseamna neaparat ca toate inregistrarile au fost citite si introduse in lista!**

**Mecanisme folosite:**

Producator-consumator pentru adaugare respectiv preluare din coada. Implementarea Producator-Consumator nu se face folosind variabile conditionale; pentru Java se va folosi mecanismul wait/notify corespunzator monitorului fiecarui obiect, iar pentru C++ se poate folosi fie un semafor fie ‘busy –waiting’ (chiar daca busy-wainting nu este o varianta foarte eficienta si prin urmare nici recomandata – intentia este de a face comparatie cu varianta cu variabile conditionale folosita la laboratorul 5).

Sincronizare la nivel de lista!!! (aceasta inseamna ca la adaugarea unei perechi se blocheaza intreaga lista -- pentru a se evita posibilitatea de a apare data-race). Creati o lista thread-safe!

Limbaj: la alegere intre Java si C++

Analiza timpului de executie pentru urmatoarele cazuri:

1. 5 tari cu cate maxim 100 de concurenti si minim 80 de concurenti in fiecare tara si 10 probleme – nu este acelasi numar de concurenti in fiecare tara!

Generarea datelor se va face random cu specificatia ca:

* Probabilitatea de a nu se rezolva o problema sa fie maxim 10% pe fiecare tara
* Probabilitatea de a fi frauda sa fie maxim 2%
  1. secvential
  2. p = 4, 6, 8, 16 si p\_r =1
  3. p = 4, 6, 8, 16 si p\_r =2

Masurarea timpului de executie include si citirea din fisiere si scrierea in fisierul rezultat (timp global)

Analiza: raport Tsecvential/Tparalel