Laborator 4

Obiectiv:

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

Limbaj: la alegere intre Java si C++

Note studenti

Nota la o disciplina este formata din suma notelor de la un numar de 10 proiecte – nota maxima obtinuta este 100 . Notele obtinute de fiecare student sunt incarcate intr-un fisier dedicat (proiect1.txt, proiect2.txt, ...proiect 10.txt). Ordinea in care apar notele studentilor intr-un fisier depinde de ordinea in care aceștia au trimis proiectul (deci poate fi diferita de la un proiect la altul). Proiectele nu sunt obligatorii si prin urmare este posibil ca unii studenti sa nu aiba nota pentru un anumit proiect.
Formatul datelor din in fisiere:
(ID, NotaP)
Rezultatul va fi un fisier „rezultate.txt” care va contine notele finale ale tuturor studentilor.

Rezolvarea necesita crearea unei liste inlantuite cu noduri care contin ca informatie perechea *(ID, Nota)*. In aceasta lista nu vor exista doua noduri cu acelasi ID.
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, nota)* se verifica daca exista deja in lista un nod cu acest *id* :

- Daca exista atunci se aduna *nota* la *Nota*
- Daca nu exista atunci se adauga un nou nod cu informatia *(ID, NotaP)*.

Dupa adaugarea tuturor datelor din fisiere, *Lista* va conține notele finale.

Lista se va salva intr-un fisier „rezultate.txt” de catre main-thread.

Metoda A) Implementare secventiala

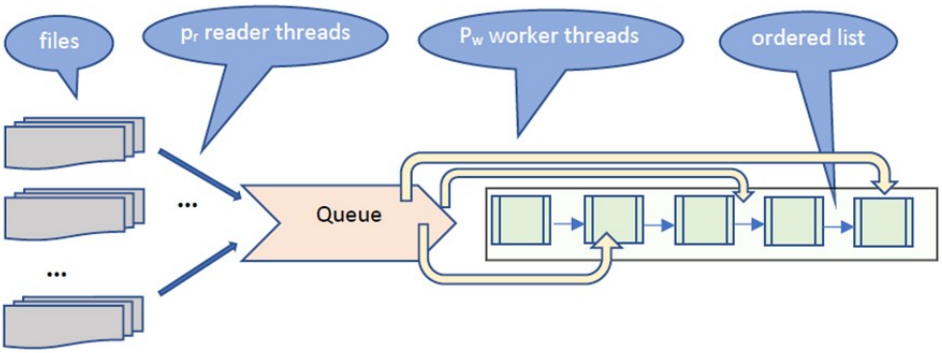
- Se citeste pe rand din fiecare fisier cate o pereche si se adauga in lista rezultat -*Lista*. La final se scriu elementele listei in fisierul “rezultat.txt”.

Metoda B1) Implementare paralela – p threaduri (nu se folosesc Executori!!!)

1. **p_r** threaduri (readers) citesc din fisiere perechi (ID, Nota) si le adauga intr-o structura de date de tip coada care este nu marginita (nu are capacitate maxim).
   (conditie – pentru structura de tip coada NU se admite folosirea unei structuri de date pentru care partea de sincronizare este deja implementata!!!)

2. Celelalte threaduri p_w= (p-p_r) threaduri (workers) preiau cate o pereche din coada si o adauga in lista *Lista*.

➔ Se continua operatiile 1., 2. pana cand toate perechile, din toate fisierele, sunt adaugate la lista *Lista.*



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,

-pentru C++ se poate folosi fie un semafor, fie 'busy –waiting' (chiar daca busy-wainting nu` este o varianta recomandata in general– intentia este de a face comparatie cu varianta cu variabile conditionale care se va folosi 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!

Testare

Se genereaza date random pentru un numar de studenti=200. Intr-un fisier vor fi minim 80 de note.

Analiza timpului de executie pentru urmatoarele cazuri:

i. secvential
ii. p = 4, 8, 16 si p_r =1
iii. p = 4, 8, 16 si p_r =2

Masurarea timpului de executie include si citirea si scrierea in fisierul rezultat.

Analiza: raport Tsecvential/Tparalel