# PPD – Laborator 4 – Documentație

## Cerință

Laborator 4

Deadline: saptamana 9

Obiectiv:

· Intelegerea/aprofundarea sablonului “producator-consumator”

· Intelegerea/aprofundarea sincronizarii conditionale

· Intelegerea/aprofundarea excluderii mutuale (granularitatea sectiunilor critice)

Se considera n polinoame reprezentate prin lista de monoame (reprezentare: lista inlantuita ordonata dupa exponentii monoamelor).

Se cere adunarea polinoamelor folosind o implementare multithreading (p threaduri).

Polinoamele se citesc din fisiere – cate un fisier pentru fiecare polinom:

- un fisier contine informatii de tip (coeficient, exponent) pentru fiecare monom al

unui polinom,

(conditie: fisierele nu contine monoame cu coeficient egal cu 0)

Metoda:

A) paralela

1. Se creeaza o lista inlantuita - L corespunzatoare unui polinom nul.

2. Primul thread citeste cate un monom si il adauga intr-o structura de date de tip coada.

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

3. Celelalte threaduri preiau cate un monom din coada si il aduna la polinomul reprezentat in lista L.

è Se continua operatiile 2., 3. pana cand toate monoamele, din toate fisierele, sunt adunate la lista L.

4. Rezultatul obtinut in lista L se scrie intr-un fisier rezultat

(conditie: fisierul nu contine monoame cu coefficient egal cu 0)

Sincronizare la nivel de lista!!!

Limbaj: la alegere intre Java si C++

Metoda B) Rezolvare secventiala

· Threadul unic citeste si adauga in lista rezultat.

Analiza timpului de executie pentru urmatoarele cazuri:

1) 10 polinoame fiecare cu gradul maxim 1000 si cu maxim 50 monoame

a. p = 4, 6, 8

b. secvential

2) 5 polinoame fiecare cu gradul maxim 10000 si cu maxim 100 monoame

a. p = 4, 6, 8

b. secvential

Fisierele input se creeaza prin generare de numere aleatoare!

## Implementare (Java)

* În package-ul data  
  + Clasa PolynomialGenerator care primește PARAMETRI maxGrade (gradul maxim al polinoamelor), maxSize (numărul maxim de monoame într-un polinom), polynomialNumber (numărul de polinoame), cu METODELE createFile (generează un fișier cu numele “polynomial[X].txt”, unde [X] este un număr de la 1 la polynomialNumber care conține un polinom) și generatePolynomials (generează polynomialNumber de astfel de fișiere);
  + Package-ul files în care se găsesc fișierele generate.
* În package-ul Lab4\_ListSync  
  + Clasa Lab4\_ListSync cu METODELE validate (verifică dacă fișierul rezultat de execuția paralelă conține aceleași date ca și cel rezultat de execuția secvențială) și main (execută programul fie secvențial, fie paralel, fie de creare și afișează timpul).
  + Clasa Parallel cu:
    - PARAMETRI result (un dicționar cu cheia si valoarea întregi reprezentând un polinom), queue (o coadă de tip LinkedList ce conține polinoamele), noPolynomials și noPolynomialsAdded (întregi care reprezintă numărul de polinoame respectiv numărul de polinoame adăugate în queue);
    - Clasa MyReaderthread cu METODA run (citește câte un polinom și îl adaugă în queue);
    - Clasa MyComputerThread cu METODA run (preia câte un polinom din queue și îl adaugă corespunzător la result);
    - METODA solve (creează threadurile și le pornește execuția după inițializarea parametrilor, după care afișează resultatul într-un fișier nou „resultParallelLab4.txt”).
* În package-ul sequencial  
  + Clasa Sequencial cu METODA solve (primește numărul de polinoame și, după citirea fișierelor generate, scrie rezultatul într-un fișier nou „result.txt”).

## Ideea algoritmului

* În metoda solve crează 1 thread de citire și p-1 threaduri de calcul. Metoda apelează metoda .start() pentru fiecare dintre threaduri și așteaptă terminarea lor cu .join().
* Un thread, din fișierele ce conțin polinoamele (sub forma „exponentˬcoeficient”) se extrag, sub formă de dicționar, polinoamele și se adaugă în coadă.
* Celelalte threaduri încearcă să scoată din coadă dacă aceasta nu este goală câte un polinom și adaugă coeficienții la cei din dicționarul ce conține rezultatul (result). Threadurile fac sincronizare (blochează) pe obiectul result.
* Execuția primului thread se oprește când s-a citit tot din toate fișierele.
* Execuția celorlalte threaduri se oprește când se extrage un polinom cu valoarea null (deci coada este goală) și s-au citit toate polinoamele (variabila noPolynomialsAdded ne informează câte polinoame s-au citit și adăugat)
* La final, dicționarul rezultat (result) este complet și se transformă într-un dicționar ordonat, urmând să se afișeze toate perechile cheie-valoare în fișierul „resultParallelLab4.txt”.

## Analiza execuției

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nr. Threaduri | Nr Polinoame | | Grad Maxim | Nr Maxim Monoame | Timp |
| secvential | 10 | 1000 | | 50 | 7176220 |
| 4 | 10 | 1000 | | 50 | 8189920 |
| 6 | 10 | 1000 | | 50 | 9116720 |
| 8 | 10 | 1000 | | 50 | 9812340 |
| secvential | 5 | 10000 | | 100 | 7037420 |
| 4 | 5 | 10000 | | 100 | 6614120 |
| 6 | 5 | 10000 | | 100 | 8915220 |
| 8 | 5 | 10000 | | 100 | 11030700 |

## Observații

* Timpul execuției paralele este fluctuant.
* De obicei, timpul cel mai rapid este obținut în cazul utilizării a 4 threaduri.
* Cu cât datele sunt mai mari, cu atât execuția paralelă se îmbunătățește