Laborator 4

**Cerința:**

Laborator 4

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.

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

Consideratii generale:

- reprezentarea unui polinom in memorie: lista inlantuita (1 nod=1monom) ordonata dupa exponentii monoamelor cu urmatorul INVARIANT (predicat adevarat la orice moment al executiei) de reprezentare:

-monoamele sunt ordonate dupa exponenti

-nu se pasteaza in lista monoame cu coeficient 0;

- nu exista doua noduri (monoame) cu acelasi exponent

- 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,

- fisierele input se creeaza prin generare de numere aleatoare.

(Conditie: fisierele nu contin monoame cu coeficient egal cu 0!)

Rezolvare:

Se porneste prin crearea unei liste inlantuita - L corespunzatoare unui polinom nul.   
In final aceasta lista va continue polinomul rezultat.

Metoda A) Implementare secventiala

* Se citeste pe rand din fiecare fisier cate un monom si se adauga in lista rezultat -L (atentie – invariantul trebuie sa ramana adevarat dupa fiecare adaugare de monom).

Metoda B) Implementare paralela – p threaduri

1. 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!!!)

1. Celelalte threaduri preiau cate un monom din coada si il aduna la polinomul reprezentat in lista L.
   * Se continua operatiile 1., 2. pana cand toate monoamele, din toate fisierele, sunt adunate la lista L.
2. Primul thread scrie rezultatul obtinut in lista L intr-un fisier rezultat

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

**Productor-consumator**

**Sincronizare la nivel de lista!!!**

Limbaj: la alegere intre Java si C++

Analiza timpului de executie pentru urmatoarele cazuri:

1. 10 polinoame fiecare cu gradul maxim 1000 si cu maxim 50 monoame
   1. p = 4, 6, 8
   2. secvential
2. 5 polinoame fiecare cu gradul maxim 10000 si cu maxim 100 monoame
   1. p = 4, 6, 8
   2. secvential

Analiza: raport Tsecvential/Tparalel

**Proiectare:**

Pentru ambele variante (secvențială și paralelă) am implementat clasele **Node** și **MyList**. **Node** reprezintă un monom, și este tipul de date al elementelor din lista de tip **MyList**. Conține un coeficient, un exponent, și următorul nod. **MyList** conține **head**, ce reprezintă primul element al listei, și **length**, lungimea sa.

**MyList** are următoarele metode:

* insertNode(node: Node) – inserează un nod în listă
* getNodeByEx(ex: Integer) – obține nodul din listă cu exponentul dat
* addNodeCoefficient(node: Node) – adaugă coeficientul nodului dat la coeficientul corespondent din listă, daca există; dacă nu există, îl adaugă folosind insert
* writeToFile – scrie lista în fișierul rezultat

În plus pentru varianta paralelă, am implementat structura pentru coada de monoame, **MyQueue**. Aceasta conține un **ArrayList** de tip **Node**, ce conține monoamele ce vor fi procesate de consumeri.

**MyQueue** are următoarele metode:

* firstElement() – returnează primul nod din coadă
* enqueue(node: Node) – inserează nodul dat în coadă
* dequeue() – scoate din coadă primul nod
* isEmpty() – verifică dacă coada este goală

Pentru varianta secvențială, pur și simplu se iau toate monoamele și se tot adaugă unei liste rezultat, folosind metoda **addNodeCoefficient(node)**.

Pentru varianta paralelă, thread-ul cu id 0 citește din fișiere monoamele și le adaugă în coadă (are rol de producer). Celelalte **p – 1** thread-uri vor fi consumerii, vor primi ca argumente lista rezultat și coada de noduri, și vor tot lua elemente din coadă, cât timp mai există elemente în ea. Dupa ce aceste elemente sunt adăugate în lista rezultat, sunt scoase din coadă.

Pentru sincronizarea thread-urilor, am pus keyword-ul **synchronized** la metodele thread-urilor (adică cea care citește din fișier, și cea care adaugă la lista rezultat).

**Timpi de execuție:**

|  |  |  |  |
| --- | --- | --- | --- |
| Tip Matrice | Nr threads | Timp executie | Raport |
| P0 (7 polinoame mici) | 1 | 28.01447 |  |
|  | 4 | 22.94155 | 1.2211 |
|  | 6 | 23.17608 | 1.2088 |
|  | 8 | 23.27495 | 1.2038 |
|  | 16 | 23.48461 | 1.1930 |
| P1 (1000 grad maxim - 50 monoame) | 1 | 44.78873 |  |
|  | 4 | 38.49074 | 1.1634 |
|  | 6 | 37.96053 | 1.1798 |
|  | 8 | 38.7081 | 1.1547 |
|  | 16 | 38.38202 | 1.1668 |
| P2 (10000 grad maxim - 100 monoame) | 1 | 42.37569 |  |
|  | 4 | 39.00089 | 1.0864 |
|  | 6 | 39.54659 | 1.0715 |
|  | 8 | 39.03126 | 1.0856 |
|  | 16 | 39.76552 | 1.0660 |
| P3 (1000 grad maxim - 100 monoame) | 1 | 58.52813 |  |
|  | 4 | 50.05067 | 1.1693 |
|  | 6 | 48.58742 | 1.2046 |
|  | 8 | 49.1715 | 1.1901 |
|  | 16 | 48.43299 | 1.2084 |

**Analiză timp:**

Rapoartele de timp Tsecvential/Tparalel se pot observa în tabelul de timpi de mai sus, pe ultima coloană.

**Diagramele claselor:**

