**Documentație Tema 5 Baciulescu Raul, gr. 236**

**Enuntul problemei:**

Aceeasi problema ca si la Laborator 4 dar cu urmatoarele modificari:

* Dintre cele p threaduri p1 sunt cititoare si p2 sunt de tip ‘worker’ (preiau din coada si adauga in lista rezultat) – p=p1+p2
* Coada in care se adauga monoamele citite are o capacitate maxima = MAX
* Implementarea pentru producator consumator (writer/reader) trebuie sa foloseasca mecanisme de tip wait notify (nu busy waiting)
* **Sincronizarea operatiile cu lista inlantuita se fac la nivel de nod nu pentru intreaga lista!**
  + Sincronizarile se vor face folosind variabile de tip Lock asociate cu fiecare nod.
  + Daca rezulta un nod cu coeficient nul acest nod se va sterge din lista!

Observatie: Este obligatoriu sa se foloseasca sablonul ‘producator-consumator’ pentru legatura dintre threadurile ‘reader’ si cele de tip ‘worker’ --- wait|notify.

**Enuntul problemei lab4:**

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.

**Detalii de implementare:**

Graphical user interface, text, application, chat or text message

Description automatically generated

Am folosit clasele utilitare CreateFile, care creaza un fisier de output, daca acesta nu exista, PolynomialGenerator care genereaza un polinom in functie de mai multi parametri si clasa Constants care contine anumite variabile cheie programului. PolynomialGenerator genereaza polinoame random in functie de variabilele maxMono, numberOfPolynomials si maxDegree si le scrie in fisiere separate. Un fisier de polinom are urmatorul format

value1 exponent1

value2 exponent2

…

valuen exponent

MyComparator, primeste doua path-uri ale unor fisiere si le compara.

A screenshot of a computer

Description automatically generated with medium confidence

Pentru implementarea secventiala, am folosit clasa Sequential, care prin functia process citeste pe rand din fiecare fisier de polinoame, si adauga fiecare monom la lista MyLinkedList.

Clasa MyLinkedList contine implementarea unei liste inlantuite care incepe de la nodul first.

La fiecare adaugare a unui nod verific daca exponentul acestuia exista in lista. Daca exista ii caut exponentul corespunzator si cresc valoarea, daca nu creez un nod nou, pastrand lista ordonata descrescator. Daca valoarea unui nod ajunge la 0, sterg nodul. Sincronizarea este facuta printr-un lock, la nivel de nod.

La adaugare, parcurg lista inlantuita cu 2 noduri, nodul current si nodul de inainte pentru a putea face stergerea mai usor. De fiecare data dau unlock la nodul precedent la parcurgere. Nodul current va avea lock, pentru a ne putea deplasa in lista si pentru a putea modifica valoarea acestuia.

Pentru implementarea paralela am folosit sablonul Producer Consumer. Clasa Parallel este responsabila de crearea thread-urilor, listei inlantuite cu rezultatul si a coadei. Coada MyQueue este o coada sincronizata, umpluta de Producer. Producer citeste fiecare fisier de polinoame si adauga monoamele sincronizat. Sincronizarea este facuta adaugand cuvantul synchronized la antetul functiilor din MyQueue. Functia pop() din MyQueue contine si un apel la wait(), atunci cand coada este goala. Pop() isi continua executia atunci cand este adaugat ceva in coada si se apeleaza functia notifyAll(). Ca sa marchez ca nu mai exista nimic de executat de consumer, adaug null-uri in coada, atatea null-uri cate consumer am. Ca sa aflu cand trebuie adaugate null-urile in coada, am un contor in coada “notFinished”. De fiecare data cand un Producer termina executia apeleaza functia insertNull, care decrementeaza contorul notFinished(notFinished la inceput e egal cu numarul de Producere). Cand notFinished este 0, adica cand toate producerele au terminat, pot adauga null-uri in coada. Cand un consumer citeste un null, inseamna ca acel consumer trebuie sa se opreasca pentru ca nu mai exista monoame de procesat.

De asemenea cand coada este plina astept cu wait() pana cand se consuma ceva din coada, marchez cu notifyAll, cand consum.

Clasa Parallel imparte numarul de thread-uri egal, adica daca am 4 thread-uri, vor fi 2 Producere si 2 Consumere. Daca am 7 thread-uri vor fi 3 Producere si 4 Consumere. Functia run din Parallel creaza thread-urile si le da join la finalul executiei.

**Teste:**

1. 10 polinoame fiecare cu gradul maxim 1000 si cu maxim 50 monoame si MAX=20
   1. p = 4:

p1 = 2: 81.45437

p1 = 3: 90.12467

* 1. p = 6:

p1 = 2: 80.2574

p2 = 3: 83.37585

* 1. p = 8:

p1 = 2: 79.45251

p2 = 3: 82.90701

* 1. secvential: 54.1

1. 5 polinoame fiecare cu gradul maxim 10000 si cu maxim 100 monoame si MAX=30
   1. p = 4:

p1 = 2: 66.79811

p1 = 3: 75.2035

* 1. p = 6:

p1 = 2: 62.76609

p2 = 3: 64.64001

* 1. p = 8:

p1 = 2: 63.95851

p2 = 3: 67.9051

* 1. secvential: 52.4
* se observa ca procesul de creare a thread-urilor ingreuneaza foarte mult aplicatia.
* cele mai bune executii sunt cele cu cel mai mic numar mic de Produceri.
* cele mai bune executii sunt cu 6 si 8 threaduri, 2 Produceri
* timpul secvential este mai bun decat cel concurent
* comparativ cu laboratorul 4 avem timpi mai mari
* un motiv pentru care avem aceste rezultate este faptul ca fisierele sunt mici, functiile de generare a polinoamelor creaza polinoame cu maxim un numar de monoame(asta insemnand ca daca maximul este de 50 de monoame, functia poate creea polinoame de 5 6 monoame, sanse foarte mici sa avem 50 de monoame).
* un alt motiv este puterea procesorului care nu este ingreunat de sarcini secventiale, pentru ca este suficient de puternic ca sa le execute repede.