Programare paralela si distribuita – Laborator 5

Munteanu Bianca-Stefania

Grupa 235/1

12.12.2022

**Analiza cerințelor**

* **Tema**: Se considera **n** polinoame reprezentate prin lista de monoame. Se cere adunarea polinoamelor folosind o implementare multithreading (p threaduri).

**Se aduc urmatoarele modificari fata de laboratorul trecut:**

* 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.

* **Reprezentare:** - 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

* **Datele de intrare**
* 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 dar nu sunt ordonate dupa exponent!)

* **Implementare:** Java
* **Testare:** masurati timpul de executie pentru

1. 10 polinoame fiecare cu gradul maxim 1000 si cu maxim 50 monoame si MAX=20
   1. p = 4, 6, 8 ; si p1=2 , p1=3
   2. secvential
2. 5 polinoame fiecare cu gradul maxim 10000 si cu maxim 100 monoame si MAX=30
   1. p = 4, 6, 8 ; si p1=2 , p1=3
   2. secvential

Fisierele trebuie creat anterior prin adaugare de numere generate aleator. Toate rularile trebuie executate cu acelasi fisier.

* **Postconditie:** Fisierul “result.txt” va contine suma celor n polinoame.
* **Analiza:** comparatia performantei –Lab 4) versus Lab 5)

**Proiectare**

**Reprezentare**

Cu ajutorul clasei **Monomial** vom reprezinta nodurile listei inlantuite. Fiecare obiect de acest tip va retine coeficientul, exponentul, legatura catre urmatorul monom, legatura catre monomul anterior si va avea un lock asociat.

class Monomial {  
 protected int coefficient;

protected int exponent;

protected Monomial next;

protected Monomial prev;

protected ReentrantLock nodeLock;

}

**PolyLinkedList** reprezinta o lista inlantuita si va contine referinta catre primul monom (nod) al listei si va avea asociat un lock pe care il vom utiliza doar la adaugarea primului element in lista.

public class PolyLinkedList {

private Monomial head;  
  
 private ReentrantLock listLock = new ReentrantLock();

**MyQueue** este o coada sincronizata pe care o voi utiliza in varianta paralelizata a programului. Contine dimensiunea curenta, referinta catre primul si ultimul monom.

public class MyQueue {

private int size;

private Monomial head;

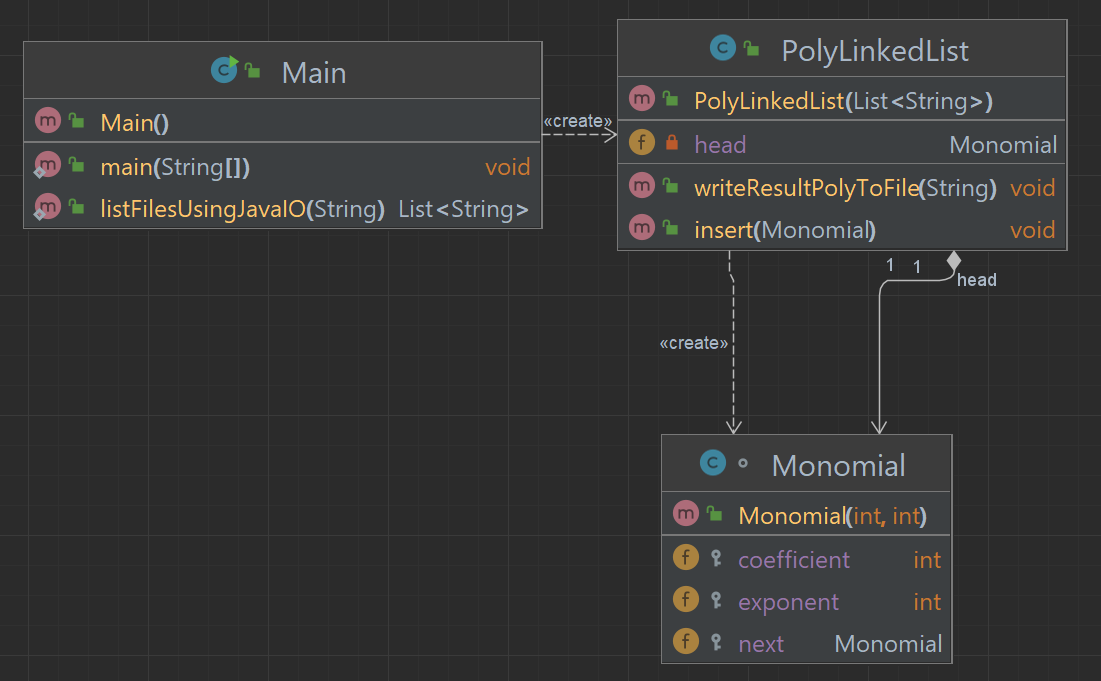
private Monomial tail;

}

**Implementare**

Am generat datele de test intr-un program separat si am copiat fisierele in cele doua proiecte corespunzatoare variantei secventiale, respectiv paralelizate.

**Metoda A) - Implementare secventiala**



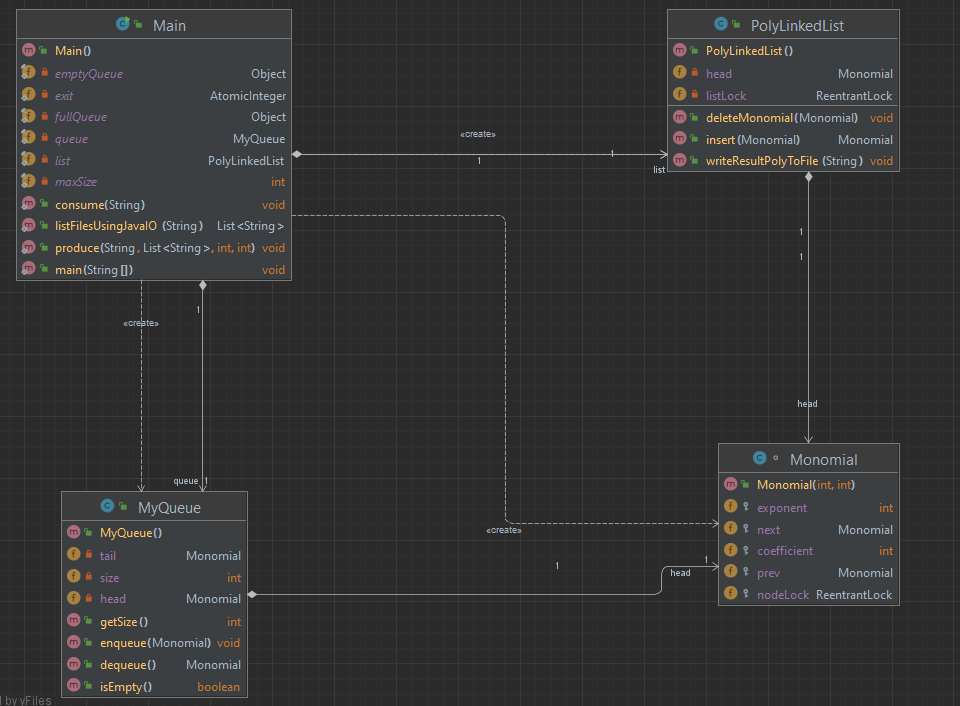
Pasi:

1. Adaug intr-o lista numele fisierelor in care se afla polinoamele ce trebuie adunate
2. Pornesc timerul
3. In constructorul clasei PolyLinkedList, parcurg numele fisierelor primite ca parametru, le iterez pe rand linie cu linie si adaug fiecare monom in lista rezultat -> vezi pasul 4 (cu respectarea conditiei ca invariantul ramane adevarat dupa fiecare adaugare de monom)
4. Inserare in lista inlantuita ordonata:

* Daca lista este goala, atunci initializez primul element
* Daca exponentul primului element >= exponentul monomului curent
  + Daca au acelasi coeficient, maresc coeficientul
  + Altfel, adaug monomul current la inceputul listei
* Altfel, caut pozitia pe care trebuie sa inserez monomul current
  + Daca au acelasi coeficient, maresc coeficientul
  + Altfel, adaug monomul current pe pozitia gasita

1. Opresc timerul
2. Afisez pe ecran timpul de executie al programului

**Metoda B) - Implementare paralela – p threaduri**



Pasi:

1. Primesc ca argument p (nr de threaduri) si p1 (nr de producatori)
2. Calculez p2, adica numarul de consumatori ca diferenta intre p si p1
3. Adaug intr-o lista numele fisierelor in care se afla polinoamele ce trebuie adunate
4. Pornesc timerul
5. Distribui uniform fisierele relativ la numarul de producatori
6. Creez p1 threaduri producator

Fiecare thread producator:

* + Primeste ca parametri numele threadului, lista completa de fisiere si 2 pozitii ‘start’ si ‘end’, reprezentand intervalul de fisiere din care trebuie sa citeasca polinoamele
  + citeste cate un monom
  + daca coada e plina, asteapta pana cand consumatorii elibereaza un loc si notifica producatorii
  + adauga monomul intr-o structura de date de tip coada si notifica consumatorii
  + dupa ce se termina cititrea, threadul 0 isi incheie executia si mai notifica o data consumatorii pentru a aduna ultimele elemente ramase in coada

1. Creez p2 threaduri consumator

Fiecare thread consumator:

* + primeste ca parametru numele threadului
  + daca coada e plina, dar inca nu s-au terminat de citit monoamele din fisier, asteapta
  + daca coada e plina, si s-au terminat de citit monoamele din fisier, consumatorul isi incheie executia
  + preia cate un monom din coada, il aduna la polinomul reprezentat in lista si notifica threadurile

1. Pornesc threadurile
2. Astept threadurile
3. Primul thread scrie rezultatul obtinut in lista L intr-un fisier rezultat
4. Opresc timerul
5. Afisez pe ecran timpul de executie al programului

La finalul executiei celor 2 metode, compar intr-un nou program fisierele rezultat, pentru a ma asigura ca au aceleasi elemente.

public static long filesCompareByByte(Path path1, Path path2) throws IOException

**Detaliere sincronizare**

In clasa **Main:**

Spre deosebire de laboratorul trecut, in clasa Main retin 2 obiecte ‘fullQueue’ si ‘emptyQueue’, al caror singur scop este sa ajute la sincronizarea din metodele produce() si consume().

private static final Object *fullQueue* = new Object();

private static final Object *emptyQueue* = new Object();

Atunci cand coada este plina, producatorii asteapta dupa obiectul ‘fullQueue’, iar consumatorii notifica imediat dupa ce elibereaza un loc din coada.

Producator:

synchronized (*fullQueue*) {  
 while (*queue*.getSize() == *maxSize*) {  
 *fullQueue*.wait();  
 }  
 }

Consumator:

if (!*queue*.isEmpty()) {  
 Monomial node = *list*.insert(*queue*.dequeue());  
 synchronized (*fullQueue*) {  
 *fullQueue*.notifyAll();  
 }  
}

Asemanator, atunci cand coada este goala, consumatorul asteapta dupa obiectul ‘emptyQueue’, iar producatorii notifica imediat dupa ce adauga un element in coada.

Producator:

synchronized (*emptyQueue*) {  
 *queue*.enqueue(nod);  
 *emptyQueue*.notifyAll();  
}

Consumator:

synchronized (*emptyQueue*) {  
 while (*queue*.isEmpty() && *exit*.get() > 0)  
 *emptyQueue*.wait();  
}

Pentru a anunta consumatorii ca producatorii au terminat de citit polinoamele, voi folosi o variabila exit, ce va avea initial valoarea p1 (adica nr total de producatori) si se va decrementa odata ce fiecare producator isi termina executia. Astfel, cand va ajunge la 0, consumatorii sunt notificati.

private static AtomicInteger *exit*;

In clasa **PolyLinkedList** retin un obiect ReentrantLock, pe care il folosesc doar la inserarea primului element in lista.

In rest, blocarea se face la nivel de nod, fiecare nod avand asociat un lock propriu, astfel:

* daca actualizez primul element sau adaug inaintea lui, blochez doar primul element
* altfel, continui sa parcurg lista pentru a gasi locul potrivit de inserare si de fiecare data blochez nodul curent si il deblochez pe urmatorul
* cand gasesc locul de inserare voi avea 2 noduri blocate, pe care le voi debloca la finalul inserarii

De asemenea, atunci cand in momentul inserarii rezulta un monom cu coeficientul 0, apelez functia deleteMonomial() pentru a-l sterge din lista.

current.next.nodeLock.lock();

...........................

current.prev.nodeLock.unlock();

In clasa **MyQueue** metodele de enqueue() si dequeue() sunt marcate cu synchronized.

synchronized public void enqueue(Monomial Monomial)

synchronized public Monomial dequeue(){

**Rezultate**

|  |  |  |  |
| --- | --- | --- | --- |
| Cazuri | **Nr procese (p)** | **Nr producers (p1)** | **Timp executie** (ms) |
|  |  |  |  |
| Caz 1)  nrPolinoame = 10  gradMaxim = 1000  maxMonoame = 50  MAX\_QUEUE=20 | secvential |  | 12,90942 |
| 4 | 2 | 21.5115 |
| 3 | 19.52815 |
| 6 | 2 | 23.45117 |
| 3 | 21.46059 |
| 8 | 2 | 22.60362 |
| 3 | 19.64803 |
|  |  |  |  |
| Caz 2)  nrPolinoame = 5  gradMaxim = 10000  maxMonoame = 100  MAX\_QUEUE=30 | secvential |  | 10.14872 |
| 4 | 2 | 20.12002 |
| 3 | 17.49366 |
| 6 | 2 | 22.05921 |
| 3 | 19.07395 |
| 8 | 2 | 20.9104 |
|  | 3 | 20.68299 |

**Analiza rezultatelor**

Observatii:

* Varianta secventiala este de cel putin 2 ori mai buna decat cea paralelizata deoarece gestionarea threadurilor e costisitoare pentru un set mic de date de intrare
* Timpul creste usor odata cu marirea numarului de threaduri
* Din analiza variantelor cu 2, respective 3 producatori, se constata ca cea cu 3 producatori e mai eficienta

Comparatie cu laboratorul trecut(cu un singur producator si blocare la nivel de lista) vs cel curent (mai multi producatori si blocare la nivel de nod)

* Rezultatele dpdv al timpului de executie sunt asemanatoare, insa se observa ca cele mai bune rezultate sunt cele din laboratorul curent, atunci cand avem 3 producatori

Voi reprezenta grafic rezultatele obtinute pentru fiecare varianta laboratorul trecut vs cel curent (varianta cu 3 producatori):

* **Lab4 vs Lab5 – Caz 1)**

nrPolinoame = 10

gradMaxim = 1000

maxMonoame = 50

* **Lab 4 vs Lab 5 – Caz 2)**

nrPolinoame = 5

gradMaxim = 10000

maxMonoame = 100

In continuare, voi reprezenta grafic rezultatele obtinute pentru fiecare varianta, din cadrul laboratorului curent:

* **secvential vs paralel – Caz 1)**

nrPolinoame = 10

gradMaxim = 1000

maxMonoame = 50

* **secvential vs paralel – Caz 2)**

nrPolinoame = 5

gradMaxim = 10000

maxMonoame = 100