Лунгол О.С.

КІТ-118б

Лабораторна робота 14

[Паралельне виконання. Ефективність використання](https://oop-khpi.gitlab.io/#tasks)

## Мета

* Вимірювання часу паралельних та послідовних обчислень.
* Демонстрація ефективності паралельної обробки.

## Вимоги

1. Забезпечити вимірювання часу паралельної обробки елементів контейнера за допомогою розроблених раніше методів.
2. Додати до алгоритмів штучну затримку виконання для кожної ітерації циклів поелементної обробки контейнерів, щоб загальний час обробки був декілька секунд.
3. Реалізувати послідовну обробку контейнера за допомогою методів, що використовувались для паралельної обробки та забезпечити вимірювання часу їх роботи.
4. Порівняти час паралельної і послідовної обробки та зробити висновки про ефективність розпаралелювання:
   * результати вимірювання часу звести в таблицю;
   * обчислити та продемонструвати у скільки разів паралельне виконання швидше послідовного.

Код програми

package ua.khpi.oop.Lunhol14.Container;  
  
import ua.khpi.oop.Lunhol14.Head.HDR1;  
  
import java.util.concurrent.*Callable*;  
import java.util.concurrent.*TimeUnit*;  
  
public class Threads *{* public static class MyTread1 implements *Callable<*Boolean*> {* Container*<*HDR1*>* container;  
  
 @Override  
 public Boolean call*()* throws InterruptedException *{* maxMinPayment*()*;  
 return true;  
 *}* private void maxMinPayment*()* throws InterruptedException *{  
 //TimeUnit.MILLISECONDS.sleep(2);* int max = container.getElementByIndex*(*0*)*.getPayment*()*;  
 int min = container.getElementByIndex*(*0*)*.getPayment*()*;  
 for *(*int i = 0; i < container.size*()*; i++*) {* if *(*container.getElementByIndex*(*i*)*.getPayment*()* < min*) {* min = container.getElementByIndex*(*i*)*.getPayment*()*;  
 *}* if *(*container.getElementByIndex*(*i*)*.getPayment*()* > max*) {* max = container.getElementByIndex*(*i*)*.getPayment*()*;  
 *}  
 }* System.*out*.println*(*"Max payment = " + max*)*;  
 System.*out*.println*(*"Min payment = " + min*)*;  
 *}* public void set*(*Container*<*HDR1*> container) {  
 this*.container = *container*;  
 *}  
  
  
 }* public static class MyTread2 implements *Callable<*Boolean*> {* Container*<*HDR1*>* container;  
  
  
 private void sumAvgPayment*()* throws InterruptedException *{  
 //TimeUnit.MILLISECONDS.sleep(2);* int sum = 0;  
 int avg = 0;  
 for *(*int i = 0; i < container.size*()*; i++*) {* sum += container.getElementByIndex*(*i*)*.getPayment*()*;  
 *}* avg = sum / container.size*()*;  
 System.*out*.println*(*"Sum payment = " + sum*)*;  
 System.*out*.println*(*"Avg payment = " + avg*)*;  
 *}* public void set*(*Container*<*HDR1*> container) {  
 this*.container = *container*;  
 *}* @Override  
 public Boolean call*()* throws Exception *{* sumAvgPayment*()*;  
 return true;  
 *}  
 }* public static class MyTread3 implements *Callable<*Boolean*> {* public static final int *HIGHER\_PAYMENT* = 1000;  
 Container*<*HDR1*>* container;  
  
 @Override  
 public Boolean call*()* throws Exception *{* countHigherPayment*()*;  
 return true;  
 *}* private void countHigherPayment*()* throws InterruptedException *{  
 //TimeUnit.MILLISECONDS.sleep(2);* int count = 0;  
 for *(*int i = 0; i < container.size*()*; i++*) {* if *(*container.getElementByIndex*(*i*)*.getPayment*()* >= *HIGHER\_PAYMENT) {* count++;  
 *}  
 }* System.*out*.println*(*"Number of vacancies with higher payment: " + count*)*;  
 *}* public void set*(*Container*<*HDR1*> container) {  
 this*.container = *container*;  
 *}  
 }  
}*

Threads

package ua.khpi.oop.Lunhol14.Tests;  
  
import ua.khpi.oop.Lunhol14.Container.Container;  
import ua.khpi.oop.Lunhol14.Container.Threads;  
import ua.khpi.oop.Lunhol14.Head.HDR1;  
  
import java.io.IOException;  
import java.util.concurrent.\*;  
  
public class Demo *{* public static void main*(*String*[] args)* throws Exception *{  
 //Lab13.main(new String[]{"-auto"});  
 // Lab13.main(args);* Container*<*HDR1*>* linkedContainer = new Container*<>()*;  
 HDR1 rec1 = new HDR1*()*;  
 HDR1 rec2 = new HDR1*()*;  
 HDR1 rec3 = new HDR1*()*;  
  
 rec1.setPayment*(*2600*)*;  
 rec2.setPayment*(*3700*)*;  
 for *(*int i = 0; i < 90000; i++*) {* linkedContainer.add*(*rec1*)*;  
 linkedContainer.add*(*rec2*)*;  
 *}* rec3.setPayment*(*600*)*;  
 for *(*int i = 0; i < 5000; i++*) {* linkedContainer.add*(*rec3*)*;  
 *}* rec2.setPayment*(*2500*)*;  
 linkedContainer.add*(*rec2*)*;  
 rec2.setPayment*(*500*)*;  
 linkedContainer.add*(*rec2*)*;  
 Threads.MyTread1 myTread1 = new Threads.MyTread1*()*;  
 myTread1.set*(*linkedContainer*)*;  
 Threads.MyTread2 myTread2 = new Threads.MyTread2*()*;  
 myTread2.set*(*linkedContainer*)*;  
 Threads.MyTread3 myTread3 = new Threads.MyTread3*()*;  
 myTread3.set*(*linkedContainer*)*;  
 System.*out*.println*(*"++++++++multithreading++++++++++"*)*;  
 long start2 = System.*currentTimeMillis()*;  
 *ExecutorService* executorService = Executors.*newFixedThreadPool(*3*)*;  
 *Future<*Boolean*>* future = executorService.submit*(*myTread1*)*;  
 *Future<*Boolean*>* future1 = executorService.submit*(*myTread2*)*;  
 *Future<*Boolean*>* future2 = executorService.submit*(*myTread3*)*;  
 future.get*()*;  
 future1.get*()*;  
 future2.get*()*;  
 future.cancel*(*true*)*;  
 future1.cancel*(*true*)*;  
 future2.cancel*(*true*)*;  
 executorService.shutdown*()*;  
 long finish2 = System.*currentTimeMillis()*;  
 long result2 = finish2 - start2;  
 System.*out*.println*(*"++++++++single threadingd+++++++"*)*;  
 long start = System.*currentTimeMillis()*;  
 myTread1.call*()*;  
 myTread2.call*()*;  
 myTread3.call*()*;  
 long finish = System.*currentTimeMillis()*;  
 long result = finish - start;  
 System.*out*.println*(*"single threading time: " + result / 1000.0 + " seconds."*)*;  
 System.*out*.println*(*"multithreading time: " + result2 / 1000.0 + " seconds."*)*;  
  
 *}  
}*

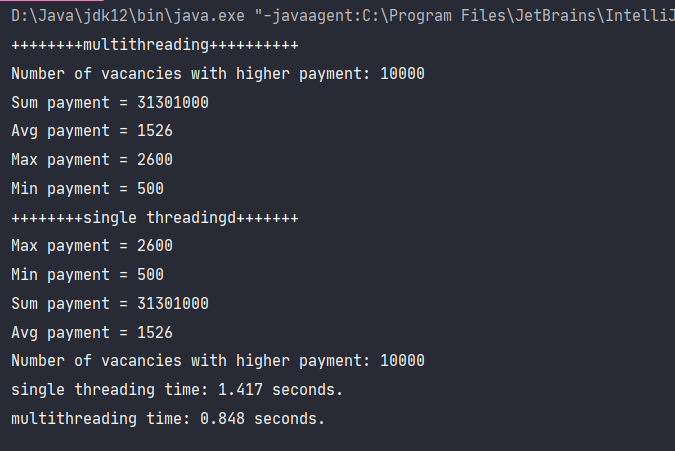


Рисунок 1 – Робота програми

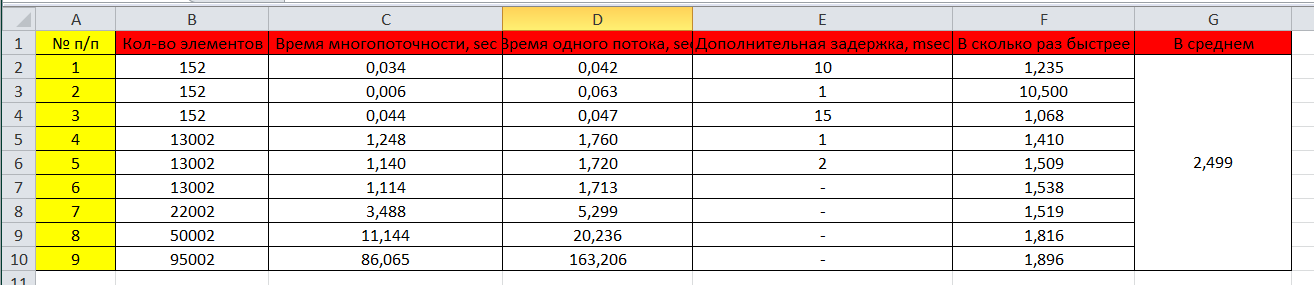


Рисунок 2 – Експеримент з різними значеннями

Висновок

Завдяки багатопоточності, досягається значна економія пам'яті,. Також досягається і економія часу, так як перемикання контексту на полегшений процес, для якого потрібно тільки змінити стек і відновити значення регістрів, значно швидше, ніж на звичайний процес.