Національний університет «Львівська Політехніка»

Кафедра Систем Штучного Інтелекту



Лабораторна робота №6

Виконав: студент КН-111

Зомчак Богдан

Перевірив:

Грабовська Н. Р.

**Львів-2019**

Тема:

Паралельне виконання. Багатопоточність. Ефективність використання.

Мета:

Ознайомлення з моделлю потоків Java. Організація паралельного виконання декількох частин програми. Вимірювання часу паралельних та послідовних обчислень. Демонстрація ефективності паралельної обробки.

import java.io.FileNotFoundException;

import java.util.Scanner;

public class Main {

static long[] times = new long[3];

static long maxRunTime;

static char first;

static char second;

static char third;

static MyContainer container = new MyContainer();

public static void main(String[] args) throws FileNotFoundException, InterruptedException {

System.out.println("Програма знаходить слова що починається на літери введені користувачем\n");

Scanner scan = new Scanner(System.in);

System.out.println("Нажміть Enter щоб розпочати");

scan.nextLine();

System.out.println("Сканую дані з файлу...");

Thread.sleep(1000);

container.setData("text.txt");

System.out.println("Сканування завершено!!");

System.out.print("Введіть максимальний час для обробки (в секундах): ");

int maxRunTimeMil = scan.nextInt();

maxRunTime = maxRunTimeMil\*1000;

scan.nextLine();

System.out.print("Введіть букви для обробки (3 букви): ");

String input = scan.nextLine();

first = input.charAt(0);

second = input.charAt(1);

third = input.charAt(2);

MyThreadShorter MyThreadShorter = new MyThreadShorter();

MyThreadLonger MyThreadLonger = new MyThreadLonger();

MyThreadEqual MyThreadEqual = new MyThreadEqual();

System.out.println("Розрочинаю обробку...");

Thread.sleep(1000);

long startMultiThreads = System.currentTimeMillis();

MyThreadShorter.start();

MyThreadLonger.start();

MyThreadEqual.start();

while (MyThreadShorter.running && MyThreadLonger.running && MyThreadEqual.running) {

Thread.sleep(10);

}

long endMultiThreads = System.currentTimeMillis();

if (MyThreadShorter.timeLimit || MyThreadLonger.timeLimit || MyThreadEqual.timeLimit) {

Thread.sleep(1000);

System.out.println("Перевищено ліміт часу виконання!");

} else {

Thread.sleep(3000);

System.out.println("Обробку завершено!");

}

System.out.println("\nНажміть Enter щоб розпочати однопотокову обробку");

scan.nextLine();

System.out.println("Розрочинаю однопотокову обробку...");

Thread.sleep(2000);

long startSingleThreads = System.currentTimeMillis();

boolean timeLimit = false;

for (int i = 0; i < container.getData().size(); i++) {

if (container.getData().get(i).charAt(0) == first) {

System.out.println("Starts by " + first + ": " +

container.getData().get(i));

try {

Thread.sleep(100);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

if ((System.currentTimeMillis()-startSingleThreads) > maxRunTime) {

timeLimit = true;

break;

}

}

long firstSingleThread = System.currentTimeMillis() - startSingleThreads;

long startSecondSingleThread = System.currentTimeMillis();

for (int i = 0; i < container.getData().size(); i++) {

if (container.getData().get(i).charAt(0) == second) {

System.out.println("Starts by " + second + ": " +

container.getData().get(i));

try {

Thread.sleep(100);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

if ((System.currentTimeMillis()-startSingleThreads) > maxRunTime) {

timeLimit = true;

break;

}

}

long secondSingleThread = System.currentTimeMillis() - startSecondSingleThread;

long startThirdSingleThread = System.currentTimeMillis();

for (int i = 0; i < container.getData().size(); i++) {

if (container.getData().get(i).charAt(0) == third) {

System.out.println("Starts by " + third + ": " +

container.getData().get(i));

try {

Thread.sleep(100);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

if ((System.currentTimeMillis()-startSingleThreads) > maxRunTime) {

timeLimit = true;

break;

}

}

long thirdSingleThread = System.currentTimeMillis() - startThirdSingleThread;

long endSingleThreads = System.currentTimeMillis();

if (timeLimit) {

System.out.println("Перевищено ліміт часу виконання!");

} else {

System.out.println("Обробку завершено!\n\n");

}

System.out.println("Щоб вивести результати нажміть Enter");

scan.nextLine();

System.out.println("Сумарний час виконання в багатопотоковому режимі: " +

Math.max(Math.max(times[0], times[1]), times[2]) + " мс.");

System.out.println("Час виконання потоку №1: " + times[0] + " мс.");

System.out.println("Час виконання потоку №2: " + times[1] + " мс.");

System.out.println("Час виконання потоку №3: " + times[2] + " мс.");

System.out.println("\nЧас виконання в однотоковому режимі: " +

(endSingleThreads-startSingleThreads) + " мс.");

System.out.println("Час виконання потоку №1: " + firstSingleThread + " мс.");

System.out.println("Час виконання потоку №2: " + secondSingleThread + " мс.");

System.out.println("Час виконання потоку №3: " + thirdSingleThread + " мс.");

}

} import java.io.File;

import java.io.FileNotFoundException;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.Scanner;

public class MyContainer implements Serializable {

private ArrayList<String> data = new ArrayList<>();

public void setData(String file) throws FileNotFoundException, InterruptedException {

File f = new File(file);

Scanner read = new Scanner(f);

read.useDelimiter("[^A-Za-z]+");

while (read.hasNextLine()) {

String word = read.next();

data.add(word);

System.out.println(word);

Thread.sleep(100);

}

}

public ArrayList<String> getData() {

return data;

}

} **class** MyThreadShorter **extends** Thread {

**public** **static** **boolean** *running* = **true**;

**public** **static** **boolean** *timeLimit* = **false**;

@Override

**public** **void** run () {

**long** startThread = System.*currentTimeMillis*();

**for** (**int** i = 0; i < Main.*container*.getData().size(); i++) {

**if** (Main.*container*.getData().get(i).charAt(0) == Main.*first*) {

System.***out***.println("Starts by " + Main.*first* + ": " +

Main.*container*.getData().get(i));

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

**if** ((System.*currentTimeMillis*()-startThread) > Main.*maxRunTime*) {

*timeLimit* = **true**;

**break**;

}

}

Main.*times*[0] = System.*currentTimeMillis*()-startThread;

*running* = **false**;

}

}

**class** MyThreadLonger **extends** Thread{

**public** **static** **boolean** *running* = **true**;

**public** **static** **boolean** *timeLimit* = **false**;

@Override

**public** **void** run () {

**long** startThread = System.*currentTimeMillis*();

**for** (**int** i = 0; i < Main.*container*.getData().size(); i++) {

**if** (Main.*container*.getData().get(i).charAt(0) == Main.*second*) {

System.***out***.println("Starts by " + Main.*second* + ": " +

Main.*container*.getData().get(i));

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

**if** ((System.*currentTimeMillis*()-startThread) > Main.*maxRunTime*) {

*timeLimit* = **true**;

**break**;

}

}

Main.*times*[1] = System.*currentTimeMillis*()-startThread;

*running* = **false**;

}

}

**class** MyThreadEqual **extends** Thread {

**public** **static** **boolean** *running* = **true**;

**public** **static** **boolean** *timeLimit* = **false**;

@Override

**public** **void** run () {

**long** startThread = System.*currentTimeMillis*();

**for** (**int** i = 0; i < Main.*container*.getData().size(); i++) {

**if** (Main.*container*.getData().get(i).charAt(0) == Main.*third*) {

System.***out***.println("Starts by " + Main.*third* + ": " +

Main.*container*.getData().get(i));

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

**if** ((System.*currentTimeMillis*()-startThread) > Main.*maxRunTime*) {

*timeLimit* = **true**;

**break**;

}

}

Main.*times*[2] = System.*currentTimeMillis*()-startThread;

*running* = **false**;

}

}

Висновок: Я ознайомився з моделлю потоків Java, організував паралельну виконання декількох частин програми.