Міністерство освіти і науки України

Національний технічний університет

«Харківський політехнічний інститут»

Кафедра «Обчислювальна техніка та програмування»

ЗВІТ

Про виконання лабораторної роботи 14

“ Multithreading. Ефективність використання”

Виконавець:

студент гр. КН-920В

Олексієнко Микита

Харків 2022

Лабораторна робота №14

**Multithreading. Ефективність використання**

**Мета**

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

**1.Вимоги**

**1.1 Розробник:**

Олексієнко Микита Віталійович

студент групи КН-920В

1. варіант

**1.2 Загальне завдання**

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.

**1.3 Прикладні задачі**

Автомобіль: марка, рік випуску, технічні характеристики у вигляді "параметр - значення" (серед яких витрата палива в міському і заміському циклі), технічний стан, ціна.

**2. Опис програми**

**2.1 Засоби ООП:**

У лабороторній роботі 14 , я використовую клас Program, який має у собі лише запуск меню, розробленого у лабороторної роботі 8. Ця работа передбачає удосконалення класу Menu, використанням багатопоточного програмування, завдяки якому обробка контейнеру працювати у рази швидше.

Попередньо оголошено клас Auto, який має поля з індивідуального завдання та методи обробки ціх полей.

**2.2 Ієрархія та структура класів:**

* Auto.java
* Container.java
* Iterator.java ( interface )
* LinkedList.java
* Menu.java
* Program.java

**2.3 Важливі фрагменти програми:**

Menu.java ( додаткові методи та класи ) :

private static void CarContainerInit() throws IOException {  
  
 container.loadFromFile("save\_lab\_14.txt");  
  
 Container containerForSingleThread = new Container();  
 containerForSingleThread.loadFromFile("save\_lab\_14.txt");  
 System.out.println(ANSI\_GREEN + "Spent time by " + Thread.currentThread().getName() + ": " + ANSI\_YELLOW +  
 + executeContainerMethods() + "ms" + ANSI\_RESET);  
  
 System.out.println("Input max thread`s working time:");  
 Scanner scanner1 = new Scanner(System.in);  
 int time = scanner1.nextInt();  
  
 System.out.println("MAIN thread -> " + Thread.currentThread().getName() + " / " + Thread.currentThread().getId());  
  
 Thread thread1 = new Thread(new Runner("sort", time));  
 thread1.start();  
  
 Thread thread2 = new Thread(new Runner("find", time));  
 thread2.start();  
  
 Thread thread3 = new Thread(new Runner("print", time));  
 thread3.start();  
  
 Scanner scanner2 = new Scanner(System.in);  
 scanner2.nextLine();  
  
 System.exit(0);  
}  
  
private static long executeContainerMethods() {  
 Date start\_time = new Date();  
  
 container.sortByPrice();  
 container.findCar();  
 container.carList();  
  
 Date end\_time = new Date();  
  
 return end\_time.getTime() - start\_time.getTime();  
  
}  
  
private static class Runner implements Runnable {  
  
 private String method = "";  
 private int time = 0;  
  
 Runner ( String method, int time ) {  
 this.method = method;  
 this.time = time;  
 }  
  
 @Override  
 public void run() {  
 Date start\_time = new Date();  
  
 switch (method) {  
 case "sort" -> {  
 System.out.println("sort thread -> " + Thread.currentThread().getName() + " / "  
 + Thread.currentThread().getId() );  
 container.sortByPrice();  
 }  
 case "find" -> {  
 System.out.println("find thread -> " + Thread.currentThread().getName() + " / "  
 + Thread.currentThread().getId());  
 container.findCar();  
 }  
 case "print" -> {  
 System.out.println("print thread -> " + Thread.currentThread().getName() + " / "  
 + Thread.currentThread().getId());  
 container.carList();  
 }  
 }  
  
 Date end\_time = new Date();  
 System.out.println(ANSI\_GREEN + "Spent time by " + Thread.currentThread().getName() + ": " + ANSI\_YELLOW +  
 + ( end\_time.getTime() - start\_time.getTime() ) + "ms" + ANSI\_RESET);  
  
 try {  
 Thread.sleep(time);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}

**2.4 Перевірки швидкості:**



Рисунок 1 – Результат послідовного виконання

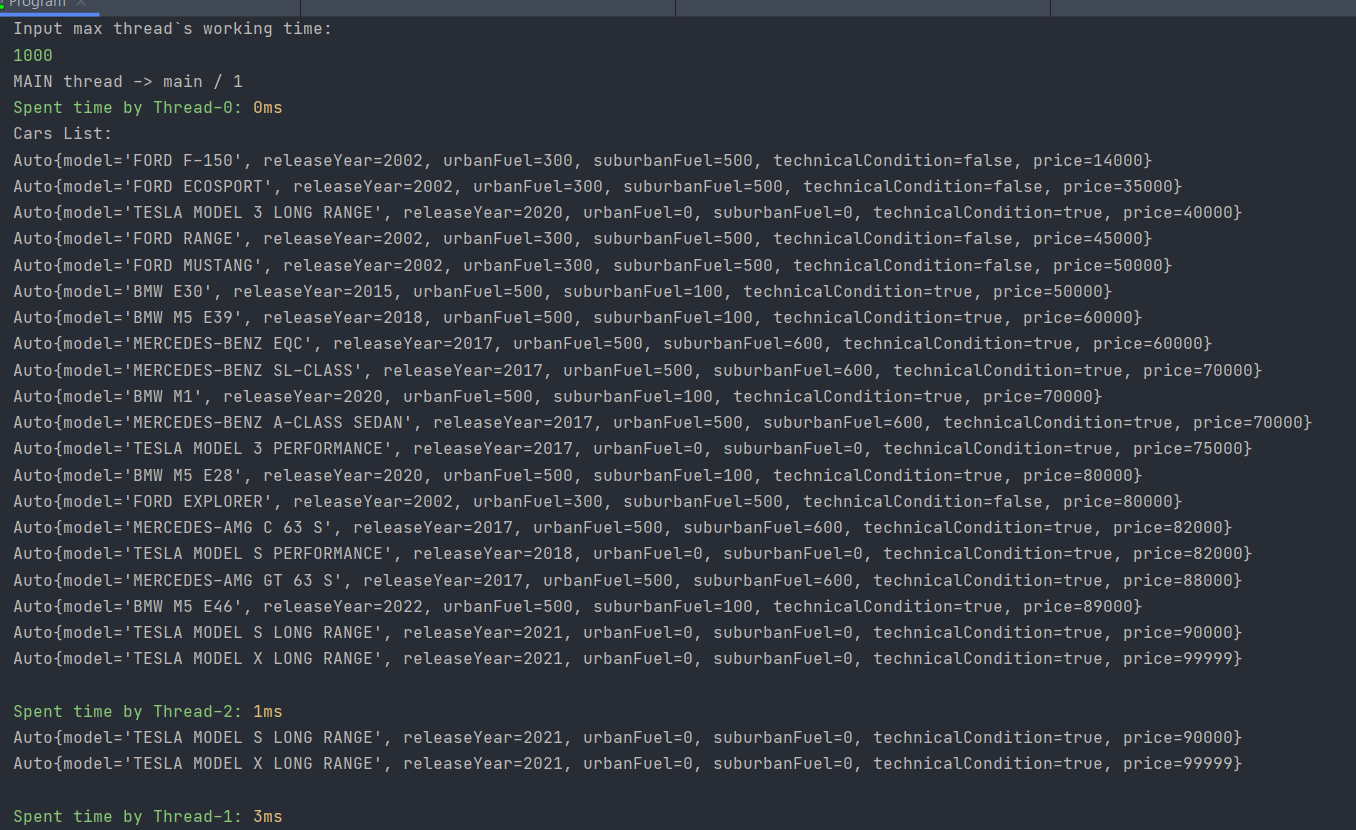


Рисунок 2 – Результат параллельного виконання

Судячи зі скриншотів вище, я можу зробити висновки, що паралельний варіант виконання у 12.3 разів швидше за свого послідовного конкурента. Саме це і є плюсом паралельного виконання – робити завдання двома руками ( у моєму випадку трьома допоміжними ) , а не однією.

Таблиця спостережень:

|  |  |  |
| --- | --- | --- |
|  | Параллельне вик. | Послідовне вик. |
| Сортування | 3ms | - |
| Знаходження елементу | 1ms | - |
| Виведення в консоль | 0.123 ms | - |
| Усього | 3ms | 37ms |

Паралельне виконання у 12,3 рази швидше.

**3. Висновок:**

* Вимірювано час паралельних та послідовних обчислень.
* Демонстровано ефективность паралельної обробки.