Тема 14. Паралельне виконання. Ефективність використання

Мета:

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

**1 ВИМОГИ**

**1.1 Розробник**

Інформація про розробника:

- Гряник Георгій Володимирович

- КІТ-119Д;

- 6 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.

2 ОПИС ПРОГРАМИ

2.1 Засоби ООП

Thread- це потік виконання в програмі. Віртуальна машина Java дозволяє додатку мати кілька потоків виконання, що працюють одночасно.

2.2 Ієрархія та структура класів

Клас “PoliceFile ” – описує поліцейську картотеку з можливістю додавати та виводити дані класу. Клас описує дані про злочинця відповідно до завданн. Клас «Date” – опису формат часу :день, місяць, рік . Створений для ергономічного запису дат відомостей про злочинця. Клас « Console\_program» - клас керування програми , створений щоб надавати користувачеві можливість керувати програмою. Клас «СontainerList» - клас-контейнер створений для зберігання даних у список. Реалізовано додавання, видалення та інші можливості для керування даними. Клас «Serializator» - клас розроблений для збереження даних контейнеру у файл. При цьому зберігання прохоже у звичайний файл та файл типу .xml. При цьому в класі реалізовано методи для відновлення даних як із звичайного файлу так із .xml файлу. Клас Console\_File розроблений для роботи із файлами розміщені в директоріях. Цей клас забезпечує можливість користувачеві обирати файл та перемінятися між директоріями. Клас «Helper» - реалізація допоміжних методів які реалізують допоміжні дії в основній програмі. Клас Obshchak – загальна область пам’яті. ProcessProcesses- клас контролювання процесів: виклик процесів та завершення.

2.3 Важливі фрагменти програми **/\*\***

**\* @author <Георгiй>**

**\***

**\*/**

@SuppressWarnings("deprecation")

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

**long** start = System.*nanoTime*();

**try** {

Serializator<ContainerList<PoliceFile>> serializator=**new** Serializator<ContainerList<PoliceFile>>();

**switch**(Obshchak.*Choice*)///пошук введеної команди

{

**case** 2:

Thread d=**new** Thread(serializator::deserializtionXML) ;

d.start();

Obshchak.*stop*=**true**;

**for**(;;)

{

Thread.*sleep*(10);

**if**(Obshchak.*stop*==**false**)**break**;

}

**if**(Obshchak.*TimeOut*!=0)

{

Thread.*sleep*(Obshchak.*TimeOut*);

d.~~stop~~();

System.***out***.print("примусово зупинено");

}

**break**;

**case** 6: //System.out.print("\n\nЗбереження даних:"+ serializator.serialization());

Obshchak.*stop* = **true**;

Thread t =**new** Thread ( serializator::serialization);

t.start();

**for**(;;)

{

Thread.*sleep*(10);

**if**(Obshchak.*stop*==**false**)**break**;

}

**if**(Obshchak.*TimeOut*!=0)

{

Thread.*sleep*(Obshchak.*TimeOut*);

t.~~stop~~();

System.***out***.print("примусово зупинено");

}

**break**;

**case** 8:

System.***out***.print("\n!!!SSS\n");

Thread sort = **new** Thread (Helper::*Sort*);

Obshchak.*stop*=**true**;

sort.start();

**for**(;;)

{

Thread.*sleep*(10);

**if**(Obshchak.*stop*==**false**)**break**;

}

**if**(Obshchak.*TimeOut*!=0)

{

sort.join(Obshchak.*TimeOut*);

sort.~~stop~~();

System.***out***.print("примусово зупинено");

}**else** sort.join();

**break**;

**case** 9:Thread reg= **new** Thread (Helper::*AppliedTask*);///time here

**long** startReg = System.*nanoTime*();

reg.*currentThread*().setName("reg");

reg.start();

**if**(Obshchak.*TimeOut*!=0)

{

Thread.*sleep*(Obshchak.*TimeOut*);

reg.~~stop~~();

System.***out***.print("примусово зупинено");

}**else** reg.join();

System.***out***.print("\n"+reg.*currentThread*().getName()+" - закінчив роботу; "+"Тривалість роботи: "+ (System.*nanoTime*() - startReg) / ***DIVIDER***+"\n");

**break**;

**case** 10:

Thread r=**new** Thread(Helper::*romove*);

Obshchak.*stop*=**true**;

r.start();

**for**(;;)

{

Thread.*sleep*(10);

**if**(Obshchak.*stop*==**false**)**break**;

}

**if**(Obshchak.*TimeOut*!=0)

{

Thread.*sleep*(Obshchak.*TimeOut*);

r.~~stop~~();

System.***out***.print("примусово зупинено");

}

**case** 11: Thread clear= **new** Thread( Obshchak.*List*::clear );///time here??

clear.start();

**if**(Obshchak.*TimeOut*!=0)

{

Thread.*sleep*(Obshchak.*TimeOut*);

clear.~~stop~~();

System.***out***.print("примусово зупинено");

}

**break**;

}

}**catch**(Exception e) { //for (int i=0;i<10;i++)System.out.println(" EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR EROR ");

System.***out***.print("\n\n\nТрапилась помилка. Але тепер все добре!!\n\n");

System.***out***.println(e);}

Thread.*currentThread*().setName("ProcessProcesses-"+Obshchak.*Choice*);

System.***out***.print("\n"+Thread.*currentThread*().getName()+" - закінчив роботу; "+"Тривалість роботи: "+ (System.*nanoTime*() - start) / ***DIVIDER***+"\n");

}/////////////////////////////////////

**private** **static** String *Name*; ///Name

**private** **static** String *Surname*; /// По-батькові

**private** **static** String *Lastname*; ///прізвище

**private** **static** **int** *index*;

**public** **static** **void** ComparisonOfParallelAndLinearSearch(ContainerList<PoliceFile> List)

{

**if** (List.size()==0) {

System.***out***.print("Список пустий");

**return** ;

}

**int** n=1;

System.***out***.print("Введіть Імя : ");

*Name*=*in*.next();

System.***out***.print("Введіть по-Батькові : ");

*Surname*=*in*.next();

System.***out***.print("Введіть прізвище : ");

*Lastname*=*in*.next();

System.***out***.print("Введіть index : ");

*index*=*in*.nextInt();

System.***out***.print("Введіть кількість ітерацій : ");

n=*in*.nextInt();

;

**long** startL = System.*nanoTime*();

**for** (**int** j=0;j<n;j++)

{

*NameSearch*();

*SurnameSearch*();

*LastnameSearch*();

*indexSearch*();

}

**double** stopL=(System.*nanoTime*() - startL) / ***DIVIDER***;

System.***out***.print("\n"+"Послідовний пошук завершнно :"+"Тривалість роботи: "+ stopL +"\n");

**long** startS = System.*nanoTime*();

Thread N = **new** Thread(Helper::*NameSearch*) ;

Thread S = **new** Thread(Helper::*SurnameSearch*) ;

Thread L = **new** Thread(Helper::*LastnameSearch*) ;

Thread I = **new** Thread(Helper::*indexSearch*) ;

**for** (**int** j=0;j<n;j++)

{ N = **new** Thread(Helper::*NameSearch*) ;

S = **new** Thread(Helper::*SurnameSearch*) ;

L = **new** Thread(Helper::*LastnameSearch*) ;

I = **new** Thread(Helper::*indexSearch*) ;

N.start();

S.start();

L.start();

I.start();

}

**try** { N.join();

S.join();

L.join();

I.join();

} **catch** (InterruptedException e) {e.printStackTrace();}

**double** stopP=(System.*nanoTime*() - startS) / ***DIVIDER***;

System.***out***.print("\n"+"Паралельний пошук завершнно :"+"Тривалість роботи: "+ stopP +"\n");

System.***out***.print("\n\n маємо наступні результати :"+

"\nПошуку елеменів у контейнері при "+n+" ітерацій отримали наступне"+

"\n\tПаралельний пошук: "+stopP+

"\n\tЛінійний пошук: "+stopL);

System.***out***.printf("\n\n Отже у даному алгоритмі при %d ітерацій швидшим обробником став %S у %f разів",

n,

(stopP<stopL)?"'Паралельний пошуk'":"'Послідовний пошуk'",

(stopP<stopL)? stopL/stopP:stopP/stopL);

}

**public** **static** **void** NameSearch()

{

**int** i=0;

**try** { Thread.*sleep*(50); } **catch** (InterruptedException e) { e.printStackTrace();}

**for** (**var** L:Obshchak.*List*)

{

**if** (L.getName().equals(*Name*))

{

System.***out***.print("Позиція вашаго елемента: "+i+"\n");

L.printTableHead();

L.show();

**return** ;

}i++;

}

// System.out.print("Ваш елемент незнайденo");

}

**public** **static** **void** SurnameSearch()

{

**int** i=0;

**try** { Thread.*sleep*(50); } **catch** (InterruptedException e) { e.printStackTrace();}

**for** (**var** L:Obshchak.*List*)

{

**if** (L.getSurname().equals(*Surname*))

{

System.***out***.print("Позиція вашаго елемента: "+i+"\n");

L.printTableHead();

L.show();

**return** ;

}i++;

} // System.out.print("Ваш елемент незнайденo");

}

**public** **static** **void** LastnameSearch()

{

**int** i=0;

**try** { Thread.*sleep*(50); } **catch** (InterruptedException e) { e.printStackTrace();}

**for** (**var** L:Obshchak.*List*)

{

**if** (L.getLastname().equals(*Lastname*))

{

System.***out***.print("Позиція вашаго елемента: "+i+"\n");

L.printTableHead();

L.show();

**return** ;

}i++;

} // System.out.print("Ваш елемент незнайденo");

}

**public** **static** **void** indexSearch()

{

**try**

{ Thread.*sleep*(50);

System.***out***.print("\n\n");

Obshchak.*List*.getNode(*index*)/\*.show()\*/;

}

**catch**(Exception e) { /\*System.out.print("Ваш елемент незнайденo")\*/;}

}

**3 ВАРІАНТИ ВИКОРИСТАННЯ**

На відмін від попередньої роботи, ця програма фіксує тривалість роботи процесів, час тривалості деяких процесів фіксується лише в самостійному режимі, тобто без спілкування з користувачем. Для демонстрації порівняння тривалості виконання завдання при послідовному та паралельному виконанні завдання.

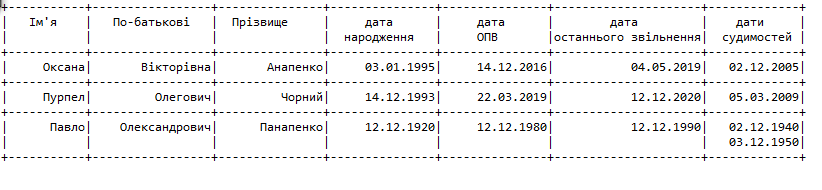


Рисунок 1. Вивід даних (частина великого списку)

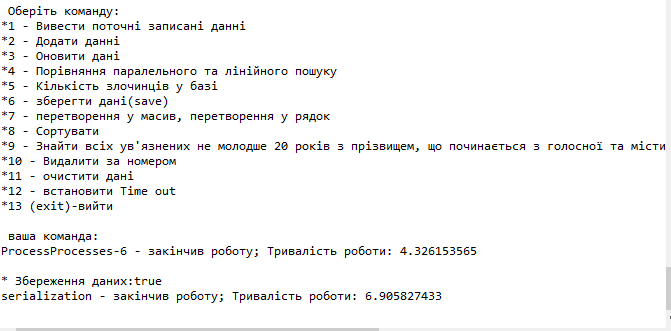


Рисунок 2. Результат роботи паралельного виконання процесів

ВИСНОВКИ

Під час виконання лабораторної роботи було набуто навички роботи з багатопоточностю, паралельною обробкою даних та ефективність використання.

Для порівняння ефективності паралельної обробки даних було створено методи пошуку елементів у списку. Завдання лінійного алгоритму у списку знайти задане ім’я, прізвище, по-батькові та індекс. Алгоритм послідовно виконує пошук кожного елементу. Для кожної ітерації фіксується час виконання та заноситься в таблицю. Час фіксується для лінійного пошуку, парольного пошуку та лінійно-паралельного пошуку .

Табл.1



Рисунок 3 графік залежності кількості ітерацій до часу виконання

За графіком ми бачимо, що як і очевидно, паралельне виконання пошуку найшвидше, а лінійний пошук є найповільнішими. Суттєву різницю ми бачимо при великій кількості операцій і очевидно, що для малих задач паралельне виконання є повільнішим , через затримку. Як ми бачимо в таблиці при одній ітерації паралельне та лінійно-паралельний пошук мають деяку відмінність, але при цьому вони виконують однакові дії. При паралельній обробці відбувається деяка затримка яка йде на запуск та завершення процесу і у такому випадку паралельна обробка може бути повільніша.