Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Лабораторна робота №5
Технології паралельних обчислень

Виконав студент групи ІТ-03: Чабан А.Є.		Перевірив: Дифучина О.Ю
		Дата:
		Оцінка:

Київ 2023

Завдання:

5.5 Завдання до комп'ютерного практикуму 5 «Застосування високорівневих засобів паралельного програмування для побудови алгоритмів імітації та дослідження їх ефективності»

1. З використанням пулу потоків побудувати алгоритм імітації багатоканальної системи масового обслуговування з обмеженою чергою, відтворюючи функціонування кожного каналу обслуговування в окремій підзадачі. Результатом виконання алгоритму є розраховані значення середньої довжини черги та ймовірності відмови. **40 балів.**
2. З використанням багатопоточної технології організувати паралельне виконання прогонів імітаційної моделі СМО для отримання статистично значимої оцінки середньої довжини черги та ймовірності відмови. **20 балів.**
3. Виводити результати імітаційного моделювання (стан моделі та чисельні значення вихідних змінних) в окремому потоці для динамічного відтворення імітації системи. **20 балів.**
4. Побудувати теоретичні оцінки показників ефективності для одного з алгоритмів практичних завдань 2-5. **20 балів.**

Хід виконання:

Клас Initializer:

```
package lab5.Systems;

import lab5.Threads.Statistic;
import lab5.Threads.Consumer;
import lab5.Threads.Producer;
import lab5.Threads.Spectator;
import lombok.AllArgsConstructor;

import java.util.concurrent.Callable;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

@AllArgsConstructor
public class Initializer implements Callable<double[]> {
    private boolean isSpectated;

    public double[] call() {
        var executor =
Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessor
s());
        var service = new Service();

        var statistic = new Statistic(service);

        // add to pool
        executor.execute(new Consumer(service));
        if (isSpectated)
            executor.execute(new Spectator(service));
        executor.execute(new Producer(service));
        executor.execute(statistic);

        executor.shutdown();

        System.out.println("System is started");

        // w8 to finish
        try {
            boolean ok = executor.awaitTermination(30,
TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return new double[]{service.calculateRejectedPercentage(),
statistic.getAverageQueueLength()};
    }
}
```

Цей клас ініціалізує і запускає систему. В методі call() створюється пул потоків,

використовуючи Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors()), це забезпечує використання всіх доступних потоків системи. Створюється об'єкт сервісу, і до пулу потоків додаються об'єкти Consumer, Producer, Statistic та Spectator (якщо встановлений). Далі викликається метод awaitTermination(), який очікує закінчення виконання всіх задач у пулі протягом 30 секунд. Якщо виконання потоків не завершено протягом цього часу, викидається виняток InterruptedException. В результаті повертається масив, де перший елемент це відсоток відхилених елементів, а другий – середня довжина черги.

Клас Service:

```
package lab5.Systems;

import java.util.ArrayDeque;
import java.util.Queue;

public class Service {
    private final int QUEUE_SIZE = 3;
    private int rejectCounter;
    private int approveCounter;
    private final Queue<Integer> queue;
    public boolean isQOpen;

    public Service() {
        approveCounter = rejectCounter = 0;
        isQOpen = true;
        queue = new ArrayDeque<>();
    }

    public synchronized void push(int item) {
        if(queue.size() >= QUEUE_SIZE) {
            rejectCounter++;
            return;
        }

        queue.add(item);
        notifyAll();
    }

    public synchronized int pop() {
        while(queue.size() == 0) {
            try {
                wait();
            } catch (InterruptedException ignored) {}
        }

        return queue.poll();
    }

    public synchronized void incrementApprovedCount() {
```

```
        approveCounter++;
    }

    public double calculateRejectedPercentage() {
        return rejectCounter / (double)(rejectCounter +
approveCounter);
    }

    public synchronized int getCurrentQueueLength () {
        return queue.size();
    }
}
```

Метод push(int item) додає елемент в чергу, якщо там є місце. Якщо немає – додавання елементу відхиляється і лічильник відхилень збільшується. Якщо черга не заповнена, елемент додається у чергу за допомогою queue.add(item). Після додавання елементу викликається notifyAll(), щоб «розбудити» всі потоки, які можуть очікувати на вільне місце в черзі.
Метод pop() - видаляє елемент з черги. Якщо черга порожня, потік викликає wait(), щоб очікувати на появу елементів у черзі. Коли елемент з'являється, він видаляється з черги за допомогою queue.poll() і повертається.
Метод incrementApprovedCount() – збільшує лічильник прийнятих елементів.
Методи calculateRejectedPercentage(), getCurrentQueueLength() – для підрахунку відхилень і довжини черги відповідно.

Клас Consumer:

```
package lab5.Threads;

import lab5.Systems.Service;
import lombok.AllArgsConstructor;

import java.util.Random;
@AllArgsConstructor
public class Consumer extends Thread {
    private final Service service;

    @Override
    public void run() {
        var random = new Random();

        while(service.isQOpen) {
            service.pop();
            try {
                Thread.sleep(random.nextInt(100));
            } catch (InterruptedException ignored) {}

            service.incrementApprovedCount();
        }
    }
```

```
}
```

Клас Consumer – реалізує логіку користувача, в методі run() в циклі користувач намагається отримати елемент з черги, після отримання елементу користувач імітує якісь дії з елементом (очікує) і збільшується лічильник схавлених елементів. Цикл працює доки черга відкрита.

Клас Producer:

```
package lab5.Threads;

import lab5.Systems.Service;
import lombok.AllArgsConstructor;

import java.util.Random;

@AllArgsConstructor
public class Producer extends Thread {
    private final Service service;

    @Override
    public void run() {
        var random = new Random();
        var startTime = System.currentTimeMillis();
        long elapsedTime = 0;

        while (elapsedTime < 10_000) {
            this.service.push(random.nextInt(100));

            try {
                Thread.sleep(random.nextInt(15));
            } catch (InterruptedException ignored) {
            }

            elapsedTime = System.currentTimeMillis() - startTime;
        }

        service.isQOpen = false;
    }
}
```

Клас Producer – реалізує логіку постачальника системи, в методі run(), в циклі виконується робота протягом 10 секунд, на кожній ітерації циклу виробник генерує новий елемент та додає його до черги. Після додавання елементу виробник робить «паузу в додаванні» на деякий час. Після проходження 10 секунд черга закривається, isQOpen = false.

Клас Statistic:

```java
package lab5.Threads;

import lab5.Systems.Service;

public class Statistic extends Thread {
    private final Service service;
    private int sumQueuesLengths;
    private int iteration;

    public Statistic(Service service) {
        this.service = service;
        sumQueuesLengths = iteration = 0;
    }

    @Override
    public void run() {
        while(service.isQOpen) {
            try {
                Thread.sleep(100);
            } catch (InterruptedException ignored) {}

            sumQueuesLengths += service.getCurrentQueueLength();
            iteration++;
        }
    }

    public double getAverageQueueLength() {
        return  sumQueuesLengths / (double)iteration;
    }
}
```

В класі Statistic – виконується логіка по збору статистика роботи системи, в методі run() збирається інформація про поточну довжину черги. В методі getAverageQueueLength() обчислюється середня довжина черги, шляхом ділення суми довжин черги на кількість ітерацій.

Клас Spectator:

```java
package lab5.Threads;

import lab5.Systems.Service;
import lombok.AllArgsConstructor;

@AllArgsConstructor
public class Spectator extends Thread {
    private Service service;

    @Override
    public void run() {
        while(service.isQOpen) {
            try {
```

```
                Thread.sleep(100);
            } catch (InterruptedException e) {}

            System.out.println("Queue size: " +
service.getCurrentQueueLength()
                    + ", fail probability: " +
Math.round(service.calculateRejectedPercentage() * 100.0) / 100.0);
        }
    }
}
```

Клас Spectator здійснює вивід всієї інформації про чергу, в методі run(). У циклі while виводиться інформація про поточну довжину черги та відсоток відмови елементів за допомогою методів getCurrentQueueLength() та calculateRejectedPercentage() об'єкта SystemService.

Завдання 1:

```
public static void task1() {
    var task = new Initializer(false);
    var results = task.call();

    printStatistic(results[0], results[1]);
}
```

Здійснюється запуск системи з обмеженою чергою, кожен функціонал системи працює в різних потоках.
Результат:

```
C:\Users\anton\.jdks\corretto-11.0.18\bin\java.exe -javaa
System is started
Fail probability: 0.85
Avg queue size: 2.77

Process finished with exit code 0
```

Завдання 2:

```
public static void task2(int systemInstancesCount) throws Exception
{
    var executor =
Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessor
s());
    var tasks = new ArrayList<Callable<double[]>>();

    for (int i = 0; i < systemInstancesCount; i++)
        tasks.add(new Initializer(false));

    List<Future<double[]>> resultList = executor.invokeAll(tasks);
```

```
    executor.shutdown();

    double totalAveragesMessages = 0, totalPercentages = 0;
    for(var result : resultList) {
        var info = result.get();

        totalAveragesMessages += info[1];
        totalPercentages += info[0];
    }

    printStatistic(totalPercentages / resultList.size(),
totalAveragesMessages / resultList.size());
}
```

В циклі створюється декілька екземплярів системи, методом invokeAll() запускаються всі завдання.

Результат:

```
C:\Users\anton\.jdks\corretto-11.0.18\bin\java.exe -j
System is started
System is started
System is started
System is started
System is started
Fail probability: 0.85
Avg queue size: 2.89
```

Завдання 3:

```
public static void task3() {
    var task = new Initializer(true);
    var results = task.call();

    printStatistic(results[0], results[1]);
}
```

Встановлюється параметр true для додавання наглядача який буде «слідкувати» за процесом роботи системи в окремому потоці.

Результат:

```
C:\Users\anton\.jdks\corretto-11.0.18\bin\j
System is started
Queue size: 3, fail probability: 0.8
Queue size: 3, fail probability: 0.83
Queue size: 3, fail probability: 0.85
Queue size: 3, fail probability: 0.85
Queue size: 3, fail probability: 0.82
Queue size: 3, fail probability: 0.84
```