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

Звіт

з комп’ютерного практикуму №5 з дисципліни
«Технології паралельних обчислень»

«Застосування високорівневих засобів паралельного програмування для побудови алгоритмів імітації та дослідження їх ефективності»

Виконав(ла) *ІП-11 Прищепа В. С.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Дифучин А. Ю.* _____
(прізвище, ім'я, по батькові)

Київ 2024

Завдання

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.

Виконання роботи

1, 2 та 3 завдання

Лістинг

Initializer.java

```
package org.example.Systems;

import org.example.Threads.Consumer;
import org.example.Threads.Producer;
import org.example.Threads.Spectator;
import org.example.Threads.Statistic;

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


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

    public Initializer(boolean isSpectated) {
        this.isSpectated = isSpectated;
    }
```

```
    public double[] call() {
        var executor =
Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
        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");

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

Service.java

```
package org.example.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();
    }
}
```

Consumer.java

```
package org.example.Threads;

import org.example.Systems.Service;
import java.util.Random;

public class Consumer extends Thread {
    private final Service service;
```

```
    public Consumer(Service service) {
        this.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();
        }
    }

}
```

Producer.java

```
package org.example.Threads;

import org.example.Systems.Service;

import java.util.Random;


public class Producer extends Thread {
    private final Service service;

    public Producer(Service service) {
        this.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;
    }
}
```

Spectator.java

```
package org.example.Threads;

import org.example.Systems.Service;

public class Spectator extends Thread {
    private Service service;

    public Spectator(Service service) {
        this.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);
        }
    }
}
```

Statistic.java

```
package org.example.Threads;

import org.example.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;
    }
}
```

Main.java

```
package org.example;

import org.example.Systems.Initializer;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class Main {

    public static void main(String[] args) throws Exception {
        //task1();
        //task2(5);
        task3();
    }

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

```
        var results = task.call();

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

    public static void task2(int systemInstancesCount) throws Exception {
        var executor =
Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
        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());
    }

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

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

    private static void printStatistic(double failProb, double avgQSize) {
        System.out.println("Fail probability: " + Math.round(failProb * 100.0) / 100.0);
        System.out.println("Avg queue size: " + Math.round(avgQSize * 100.0) / 100.0);
    }
}
```

Результат

```
System is started
Fail probability: 0.84
Avg queue size: 2.89
```

Запуск системи з обмеженою чергою, кожен функціонал системи працює в різних потоках.

```
System is started
System is started
System is started
System is started
System is started
Fail probability: 0.85
Avg queue size: 2.88
```

Створення декілька екземплярів системи.

```
System is started
Queue size: 3, fail probability: 0.67
Queue size: 3, fail probability: 0.75
Queue size: 3, fail probability: 0.77
Queue size: 3, fail probability: 0.83
Queue size: 3, fail probability: 0.77
Queue size: 2, fail probability: 0.77
Queue size: 2, fail probability: 0.78
Queue size: 3, fail probability: 0.78
Queue size: 3, fail probability: 0.79
Queue size: 3, fail probability: 0.8
Queue size: 3, fail probability: 0.81
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: 0.85
Queue size: 3, fail probability: 0.85
Queue size: 3, fail probability: 0.85
Queue size: 3, fail probability: 0.85
Queue size: 3, fail probability: 0.85
Queue size: 3, fail probability: 0.85
Queue size: 3, fail probability: 0.85
Fail probability: 0.85
Avg queue size: 2.92
```

Встановлено параметр true для додавання наглядача, який буде стежити за процесом роботи системи в окремому потоці.

4 Завдання

Побудую теоретичні оцінки показників ефективности для стрічкового алгоритму множення матриць.

Кількість комірок результуючої матриці, які обчислюються послідовно:

$$T1 = n * m.$$

Кількість комірок результуючої матриці, які обчислюються паралельно:

$$Tp = n * m / p .$$

Прискорення, одержуване при використанні p процесорів:

$$Sp = T1 / Tp = p;$$

Ефективність використання процесорів при паралельній реалізації алгоритму:

$$Ep = Sp / p = 1;$$

Висновок: Отже, було застосовано високорівневі засоби паралельного програмування для побудови алгоритмів імітації та дослідження їх ефективності.