Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт информационных технологий и анализа данных Центр программной инженерии

ОТЧЕТ

к лабораторной работе №2 по дисциплине:

|  |
| --- |
| «Технология разработки программных комплексов» |
| Распараллеливание потоков |

наименование темы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил студент |  | | | | |
| группы: | ИСТб-19-2 |  |  |  | Комогорцева Ю.В. |
| Проверил: | шифр группы  15 баллов |  | подпись |  | Фамилия ИО  Бахвалова З.А. |
|  | должность |  | подпись |  | Фамилия ИО |

Иркутск. 2022 г.

**Оглавление**

[1 Задание 3](#_Toc97332321)

[2 UML-диаграмма классов 4](#_Toc97332322)

[3 Результаты работы программы 5](#_Toc97332323)

[4 Исходный код 6](#_Toc97332324)

[5 Примеры использования параллельных алгоритмов 9](#_Toc97332325)

[Литература 10](#_Toc97332326)

**1 Задание**

Задания к лабораторным работам выполняются в соответствии с номером варианта. Номер варианта, соответствует порядковому номеру студента в списке преподавателя. Чётные варианты заданий подразумевают использование 2 (двух) потоков. Нечётные - 3 (трёх) потоков.

Все варианты заданий подразумевают использование промежуточных буферов, представляющих собой динамические массивы. Максимальный размер буферов - N чисел. N определяется для каждого варианта. Потоки, помещающие числа в буферы, следят за переполнение буферов. Потоки, извлекающие числа из буферов, могут производить данную операцию в произвольный момент времени вне зависимости от того, заполнен ли буфер полностью или нет.

**Индивидуальный вариант**

№ 24.

Значения констант и реализуемые потоками функции: **N=3**

**Первый поток** - генерирует в буфер 167 случайных чисел из интервала от 10 до 184.

**Второй поток** - извлекает числа из буфера, начиная со стоящих в позиции с большим значением индекса, и вычисляет для них значение косинуса. Результат выводится на экран.

**2 UML-диаграмма классов**

На UML диаграмме классов изображены классы двух потоков, точка входа в программу и класс Singleton.

Класс точки входа в программу создает объекты классов Thread1 и Thread2, поэтому между ними связь - зависимость.

Классы потоков Thread1 и Thread2 имеют поле instance класса Singleton и метод run, внутри которого содержится код, выполняющий функции потоков. Из-за того, что Singleton является частью классов потоков, то между ними будет связь агрегация.

Singleton – это класс реализующий паттерн проектирования синглтон. Этот класс содержит метод getInstance, который возвращает экземпляр класса. Этот медот является глобальной точкой доступа к экземпляру класса. Паттерн был применен для гарантии того, что у класса всего будет один экземпляр класса и потоки при работе с полями Singleton будут работать с одним и тем же.

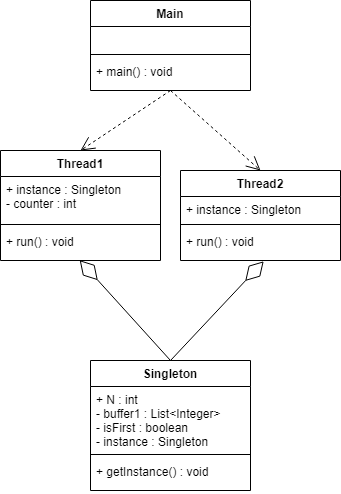


Рисунок 1 – UML диаграмма классов

**3 Результаты работы программы**

При работе программы, очередность работы потоков не управляется кодом.Из рисунка ниже видно, что потоки работают рандомно, однако буфер не переполняется.

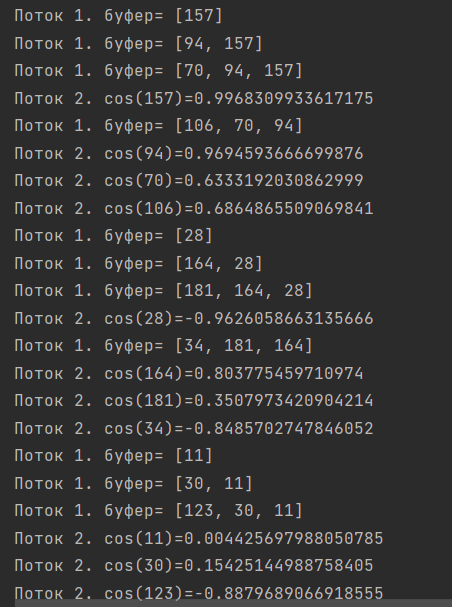


Рисунок 2 – Результат работы программы

**4 Исходный код**

Исходный код классов с комментариями, поясняющими выполнение лабораторной работы

Точка входа в программу, где создаются и запускаются два потока.

package com.company;  
  
public class Main {  
 public static void main(String[] args) throws InterruptedException {  
 Thread thread = new Thread1(); //создаем объект потока 1  
 thread.start(); //запускаем поток 1  
 Thread.*sleep*(3000); //приостанавливаем поток   
 Thread thread2 = new Thread2(); //создаем объект потока 2  
 thread2.start(); //запускаем поток 2  
 }  
}

Класс Singleton, отвечающий за гарантию, что у класса будет всего один экземпляр класса и предоставляющий глобальную точку доступа к экземпляру данного класса. Объекты этого класса используется как поля в классах потоков.

package com.company;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public final class Singleton {  
  
 final int N = 3; //размер буфера  
  
 volatile Boolean isFirst = false;  
  
 volatile List<Integer> buffer1 = new ArrayList<>(); // сам буфер  
  
 private static volatile Singleton *instance*;  
  
 public static Singleton getInstance() {  
 Singleton result = *instance*;  
 if (result != null) {  
 return result;  
 }  
 synchronized (Singleton.class) {  
 if (*instance* == null) {  
 *instance* = new Singleton();  
 }  
 return *instance*;  
 }  
 }  
}

Класс первого потока, который генерирует числа в буфер и контролирует переполнение буфера. При заполнение буфера будет вызван метод wait(), который освобождает буфер для работы другим потокам . Для генерации случайных чисел в многопоточной среде используется объект класса ThreadLocalRandom.

package com.company;  
  
import java.util.concurrent.ThreadLocalRandom;  
  
public class Thread1 extends Thread {  
  
 static Singleton *instance* = Singleton.*getInstance*();  
  
 private int counter = 0;//счетчик на кол-во сгенерируемых чисел  
  
 @Override  
 public void run() { //переопределяем метод Run  
 *instance*.isFirst=true;  
 while (counter < 167) { //цикл на генерацию чисел  
 if (*instance*.buffer1.size() >= *instance*.N) { //условие на заполненный буфер  
 try {  
 *instance*.buffer1.wait(); //освобождает монитор и переводит поток1 в состояние ожидания до тех пор, пока поток2 не вызовет метод notify()  
 } catch (Exception e) {  
  
 }  
 } else {  
 synchronized (*instance*.buffer1) { //будет блокировать доступ к коду, если буфер уже использует другой поток  
 ThreadLocalRandom random = ThreadLocalRandom.*current*(); //обьект класса предназначенный для генерации рандомных чисел в потоках  
 *instance*.buffer1.add(0, random.nextInt(184 - 10 + 1) + 10);//добавляем в буфер сгенерированное число в границах от 10 до 184  
  
 System.*out*.println("Поток 1. буфер= " + *instance*.buffer1);  
 counter++; //увеличиваем счетчик  
 }  
 }  
 }  
 *instance*.isFirst=false;  
 }  
  
}

Класс второго потока, который вычисляет косинус чисел из буфера. При этом он должен начинать работать в рандомном порядке, но не сможет работать если буфер пустой, так как он их оттуда извлекает. Как только поток вычислил косинус числа из буфера, он удаляет это число и освобождает буфер для работы.

package com.company;  
public class Thread2 extends Thread {  
  
 static Singleton *instance* = Singleton.*getInstance*();  
  
 @Override  
 public void run() {  
 while (*instance*.buffer1.size() != 0 || *instance*.isFirst) { //пока буфер не пустой или первый поток не закончил выполнение  
  
 //если попали в цикл т к первый поток не завершил работу  
 // может оказаться что буфер пустой,значит не можем отсюда взять цифры,значит надо пропустить код после условия и вернуться в начало цикла  
 if (*instance*.buffer1.size() == 0) continue;  
  
  
 synchronized (*instance*.buffer1) { //будет блокировать доступ к коду, если буфер уже использует другой поток  
 System.*out*.println("Поток 2. cos("+*instance*.buffer1.get(*instance*.buffer1.size()-1)+")=" + Math.*cos*(*instance*.buffer1.get(*instance*.buffer1.size()-1)));//выводим результат  
 *instance*.buffer1.remove(*instance*.buffer1.size()-1); //удаляем из буфера  
 *instance*.buffer1.notify(); //возобновим выполнение потока, из которого был вызван метод wait() для буфера  
 }  
 }  
 }  
}

**5 Примеры использования параллельных алгоритмов**

* Многопоточность широко используется в приложениях с пользовательским интерфейсом. В этом случае за работу интерфейса отвечает один поток, а какие-либо вычисления выполняются в других потоках. Это позволяет пользовательскому интерфейсу не подвисать, когда приложение занято другими вычислениями.
* Многие алгоритмы легко разбиваются на независимые подзадачи, которые можно выполнять в разных потоках для повышения производительности. Например, при фильтрации изображения разные потоки могут заниматься фильтрацией разных частей изображения.
* Если некоторые части приложения вынуждены ждать ответа от сервера/пользователя/устройства, то эти операции можно выделить в отдельный поток, чтобы в основном потоке можно было продолжать работу, пока другой поток ждёт ответа.

# Литература

* 1. Эрик Фримен, Элизабет Робсон Head First Паттерны проектирования обновленное юбилейное издание, 2020. 656 с. ISBN: 978-5-4461-1034-6 Серия: Head First O’Reilly (дата обращения: 17.02.2022)
  2. Репозиторий с исходным кодом проекта // GitHub URL: https://github.com/KomogortsevaYulia/TRPK/tree/main/ParallelizationThreads (дата обращения: 01.03.2022)
  3. Паттерны проектирования: Singleton // Refactoring Guru URL: https://javarush.ru/groups/posts/2365-patternih-proektirovanija-singleton (дата обращения: 02.03.2022)
  4. UML для самых маленьких: диаграмма классов //Habr URL: https://habr.com/ru/post/511798/ (дата обращения: 03.03.2022)
  5. Взаимодействие между потоками. Методы wait(), notify(), notifyAll(). Примеры // https://www.bestprog.net/ru/2021/02/06/java-interaction-between-threads-ru/#q01 (дата обращения: 03.03.2022)