Молдавский Государственный Университет

Факультет Математики и Информатики

Департамент Информатики

Лабораторная работа №6

по курсу Java

на тему

“Создание нитей процессов (Threads)”

Выполнил: Bogdanov Iurii,

группа I2302

Проверил: Gheorghe Latul

Кишинев, 2024

**Цель работы:**

Целью данной лабораторной работы является изучение основ многопоточности в языке программирования Java. В процессе выполнения задачи разработана программа, демонстрирующая использование потоков для параллельного выполнения вычислений, а также организации работы с процессами.

**Описание задачи:**

· Реализовать программу, которая на вход принимает вектор чисел, вводимых пользователем.

· Организовать защиту от некорректного ввода данных.

· Программа должна работать с использованием двух потоков:

* Первый поток ищет максимальное значение вектора.
* Второй поток ищет минимальное значение вектора.

· Обеспечить синхронное завершение работы потоков.

· Выводить результаты выполнения потоков в процессе main.

· В начале программы вывести на экран число доступных процессоров.

**Реализация:**

#### 1. Ввод данных

Для ввода данных реализован метод inputVector, который запрашивает у пользователя размер вектора и его элементы. Защита от некорректного ввода осуществляется с помощью обработки исключений. Если вводятся неверные значения (например, нечисловые или отрицательные размеры), пользователю предлагается повторить ввод.

public static int[] inputVector() {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Введите размер вектора: ");  
 int size;  
 while (true) {  
 try {  
 size = scanner.nextInt();  
 if (size <= 0) throw new IllegalArgumentException("Размер вектора должен быть положительным.");  
 break;  
 } catch (Exception e) {  
 System.*out*.println("Неверный ввод. Пожалуйста, введите положительное целое число.");  
 scanner.next();  
 }  
 }  
  
 int[] vector = new int[size];  
 System.*out*.println("Введите элементы вектора:");  
  
 for (int i = 0; i < size; i++) {  
 while (true) {  
 try {  
 System.*out*.print("Элемент [" + i + "]: ");  
 vector[i] = scanner.nextInt();  
 break;  
 } catch (Exception e) {  
 System.*out*.println("Неверный ввод. Пожалуйста, введите целое число.");  
 scanner.next();  
 }  
 }  
 }  
 return vector;  
}

#### 2. Потоки

Каждая задача (поиск максимального и минимального значений) выполняется в отдельном потоке. Используются лямбда-выражения для упрощения создания потоков.

Thread findMaxThread = new Thread(() -> {  
 System.*out*.println("Работает первый процесс");  
 int max = *findMax*(vector);  
 System.*out*.println("Максимальное значение в векторе: " + max);  
});  
  
Thread findMinThread = new Thread(() -> {  
 System.*out*.println("Работает второй процесс");  
 int min = *findMin*(vector);  
 System.*out*.println("Минимальное значение в векторе: " + min);  
});

#### 3. Обработка потоков

После запуска потоков с помощью start, используется метод join для синхронизации выполнения. Это гарантирует, что главный поток программы дождётся завершения работы дочерних потоков.

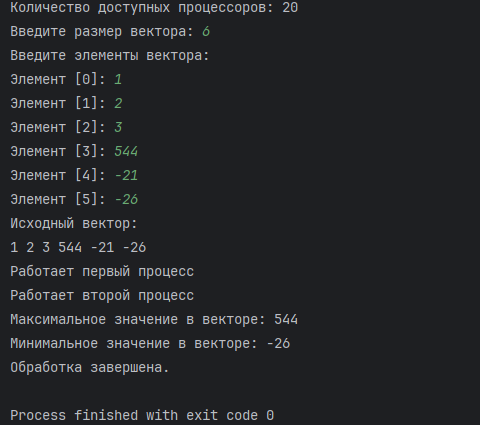
findMaxThread.start();  
 findMinThread.start();  
  
 try {  
 findMaxThread.join();  
 findMinThread.join();  
 } catch (InterruptedException e) {  
 System.*err*.println("Ошибка при ожидании потоков: " + e.getMessage());  
 }  
  
 System.*out*.println("Обработка завершена.");  
}

#### 4. Основной метод

Программа начинается с отображения числа доступных процессоров и завершает свою работу после выполнения всех потоков. Исходный вектор и результаты вычислений выводятся на экран.

public static void main(String[] args) {  
 System.*out*.println("Количество доступных процессоров: " + Runtime.*getRuntime*().availableProcessors());  
  
 int[] vector = *inputVector*();  
  
 System.*out*.println("Исходный вектор:");  
 *printArray*(vector);  
  
 Thread findMaxThread = new Thread(() -> {  
 System.*out*.println("Работает первый процесс");  
 int max = *findMax*(vector);  
 System.*out*.println("Максимальное значение в векторе: " + max);  
 });  
  
 Thread findMinThread = new Thread(() -> {  
 System.*out*.println("Работает второй процесс");  
 int min = *findMin*(vector);  
 System.*out*.println("Минимальное значение в векторе: " + min);  
 });  
  
 findMaxThread.start();  
 findMinThread.start();  
  
 try {  
 findMaxThread.join();  
 findMinThread.join();  
 } catch (InterruptedException e) {  
 System.*err*.println("Ошибка при ожидании потоков: " + e.getMessage());  
 }  
  
 System.*out*.println("Обработка завершена.");  
}

**Результат вывода программы:**



**Заключение:**

В ходе выполнения лабораторной работы я изучил основы многопоточности в Java, научился создавать и синхронизировать потоки, а также использовать их для параллельного выполнения задач. Полученные знания продемонстрировали, как можно распределять вычислительную нагрузку между процессами, эффективно используя доступные ресурсы. Программа показала стабильную работу даже при некорректном вводе, что свидетельствует о её надёжности и удобстве использования.