МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное  
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет  
имени академика С.П. Королева»  
(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики

Кафедра программных систем

**ОТЧЁТ**

по лабораторной работе №2 по дисциплине

«Моделирование и анализ параллельных алгоритмов»

Студент группы №6132-020402D В.А. Артамонов

Проверил Д.С. Оплачко

Самара 2021

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc86255005)

[2 Описание работы параллельного алгоритма 4](#_Toc86255006)

[3 Текст различных вариантов программы 5](#_Toc86255007)

[3.1 Параллельный алгоритм 5](#_Toc86255008)

[4 Результаты вычислительных экспериментов 7](#_Toc86255009)

[5 Заключение 9](#_Toc86255010)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 10](#_Toc86255011)

[ПРИЛОЖЕНИЕ А](#_Toc86255012)

[Код сигнального потока 11](#_Toc86255013)

[ПРИЛОЖЕНИЕ Б](#_Toc86255014)

[Код класса мейн 13](#_Toc86255015)

[ПРИЛОЖЕНИЕ В](#_Toc86255016)

[Код параллельного алгоритма 15](#_Toc86255017)

[ПРИЛОЖЕНИЕ Г](#_Toc86255018)

[Код последовательного алгоритма 16](#_Toc86255019)

1. Постановка задачи
2. Разработать параллельную программу, выполняющую сравнение двух файлов (по аналогии с утилитой Linux diff). Программа должна построчно сравнивать два текстовых файла и выводить на стандартный вывод номера и значения отличающихся строк. Если один из файлов длиннее другого, то программа должна вывести каждую дополнительную строку более длинного файла. Программа должна состоять из трех потоков: по одному потоку на построчное чтение каждого файла и один поток на сравнение двух считанных строк. Строки из разных файлов могут читаться одновременно, но сравнение происходит, когда считаны обе строки с одинаковыми номерами. Пока процесс, сравнивающий строки, не выполнит сравнение, процессы-читатели ждут и не читают очередные строки.

2) Измерить время работы программы для различных файлов. Файлы должны иметь различную длину строк и количество строк. Длина строк изменяется от нескольких символов до нескольких тысяч символов. Количество строк изменяется от нескольких десятков до нескольких десятков тысяч. Рекомендуется использовать не менее 10 различных значений для длин строк и количества строк в файлах при проведении экспериментов.

3) Объяснить наблюдаемые изменения длительности работы программы.

4) Сравнить длительность работы параллельной программы с длительностью работы ее последовательного аналога, осуществляющего последовательное чтение по одной строке из каждого файла и их сравнение.

5) Объяснить наблюдаемые закономерности.

6) Составить отчет по результатам работы.

1. Описание работы параллельного алгоритма

Принцип многопоточности, написанной лабораторной работы № 2 на языке Java, представлен в виде класса Thread(поток) [1]. Thread, в свою очередь, делится на три потока:

Signal – подает сигналы к запуску следующих двух потоков и потом ждет от них отклика, после чего заполняет строку символами, и так каждый раз, пока значение нумерации строк не станет «null».

Parallel – данный поток выполняет заполнение строк, каждый раз, после получения сигнала на заполнение. Поток Posled работает по такому же принципе, только последовательной реализацией алгоритма [2].

Принцип работы следующий: при помощи функции createFiles создаются 2 файла и в последствии заполняются строками, а после окончания работы могут быть удалены (настройки приложения). Далее поток Signal отправляет сигналы в параллельный и последовательный потоки. После ожидает их отклика о заполнении строки, повторяет эту процедуру, пока не закончатся строки, и при помощи метода break, прерывает работу алгоритмов. После этого выводятся номер и содержимое строк, которые различаются в созданных двух файлах, а также выводится время, затраченное на работу программы.

1. Текст различных вариантов программы
   1. Параллельный алгоритм

public class ParallelFilesComparatorThread extends Thread {

private final FileReaderThread file1;  
 private final FileReaderThread file2;

public ParallelFilesComparatorThread(FileReaderThread file1, FileReaderThread file2) {  
 this.file1 = file1;  
 this.file2 = file2;  
 }  
 public void run() {  
 String file1Line;  
 String file2Line;

boolean isEndOfFiles;  
 int currStringsNum = 1;  
 long startMillis = System.*currentTimeMillis*();

while (true) {  
 file1.runReadLine();  
 file2.runReadLine();  
 file1Line = file1.currLine();  
 file2Line = file2.currLine();  
 isEndOfFiles = file1Line == null & file2Line == null;  
 if (isEndOfFiles) {  
 break;  
 }  
 if (!Objects.*equals*(file1Line, file2Line)) {  
 System.*out*.println("#" + currStringsNum + ":\n" +  
 "v faile 1 stroka = " + file1Line + "\n" +  
 "v faile 2 stroka = " + file2Line + "\n");  
 }  
 currStringsNum++;  
 }  
 long finishMillis = System.*currentTimeMillis*();  
 file1.finish();  
 file2.finish();

long consumedMillis = finishMillis - startMillis;  
 System.*out*.println("Consumed millis: " + consumedMillis);  
 }  
}

Последовательный алгоритм

public class SequentialFilesComparatorThread extends Thread {  
 private final String[] filesNames;  
 public SequentialFilesComparatorThread(String[] filesNames) {  
 this.filesNames = filesNames;  
 }  
 public void run() {  
 try {  
 final Scanner file1Scanner = new Scanner(new File(filesNames[0]));  
 final Scanner file2Scanner = new Scanner(new File(filesNames[1]));  
 String file1Line;  
 String file2Line;  
 boolean isEndOfFiles;  
 int currStringsNum = 1;  
 long startMillis = System.*currentTimeMillis*();  
 while (true) {  
 if (file1Scanner.hasNextLine()) {  
 file1Line = file1Scanner.nextLine();  
 } else {  
 file1Line = null;  
 }  
 if (file2Scanner.hasNextLine()) {  
 file2Line = file2Scanner.nextLine();  
 } else {  
 file2Line = null;  
 }  
 isEndOfFiles = file1Line == null & file2Line == null;  
 if (isEndOfFiles) {  
 break;  
 }  
 if (!Objects.*equals*(file1Line, file2Line)) {  
 System.*out*.println("#" + currStringsNum + ":\n" +  
 "v faile 1 stroka = " + file1Line + "\n" +  
 "v faile 2 stroka = " + file2Line + "\n");}  
 currStringsNum++;  
 }  
 long finishMillis = System.*currentTimeMillis*();  
 file1Scanner.close();  
 file2Scanner.close();  
 long consumedMillis = finishMillis - startMillis;  
 System.*out*.println("Consumed millis: " + consumedMillis);  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
}

1. Результаты вычислительных экспериментов

Вычислительные эксперименты, представленные в данном разделе, проводились на вычислительной системе с процессором, обладающим характеристиками, представленными в таблице 1.

Таблица 1 – Характеристики вычислительной системы

|  |  |
| --- | --- |
| Модель процессора | Intel(R) Core(TM) i3-6006U |
| CPU | 2.00GHz |
| Число ядер | 4 |
| Hyper-Threading (multithreading) | Есть |

В ходе выполнения лабораторной работы были проведены измерения времени работы последовательного и параллельного алгоритмов. Результаты представлены в таблице 2.

Таблица 2 – Результаты работы алгоритмов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во строк в файле | Длина строки | Время работы последова-тельного алгоритма 𝑇П, мс | Время работы паралле-льного алгоритма 𝑇||, мс |  |
| 10 | 10 | 47 | 113 | 0,416 |
| 10 | 100 | 31 | 79 | 0,392 |
| 10 | 500 | 78 | 82 | 0,951 |
| 10 | 1000 | 47 | 112 | 0,42 |
| 10 | 1500 | 109 | 141 | 0,773 |
| 10 | 2000 | 94 | 112 | 0,839 |
| 10^3 | 10 | 78 | 166 | 0,47 |
| 10^3 | 100 | 125 | 199 | 0,628 |
| 10^3 | 500 | 172 | 272 | 0,632 |
| 10^3 | 1000 | 297 | 370 | 0,803 |
| 10^3 | 1500 | 360 | 476 | 0,756 |
| 10^3 | 2000 | 328 | 357 | 0,919 |
| 10^4 | 10 | 328 | 605 | 0,542 |
| 10^4 | 100 | 406 | 762 | 0,533 |
| 10^4 | 500 | 578 | 707 | 0,818 |
| 10^4 | 1000 | 688 | 857 | 0,803 |
| 10^4 | 1500 | 1312 | 1635 | 0,802 |
| 10^4 | 2000 | 1396 | 1486 | 0,939 |

Полученные данные можно представить в виде графиков.

На рисунке 1 представлен график зависимости времени работы последовательного алгоритма от размера файла и количества строк.

  
Рисунок 1 – Зависимость времени работы последовательного алгоритма от размера файла и количества строк

На рисунке 2 представлен график зависимости времени работы параллельного алгоритма от размера файла и количества строк.

  
Рисунок 2 – Зависимость времени работы параллельного алгоритма от размера файла и количества строк

На рисунке 3 представлен график ускорения работы программы.

Рисунок 3 – Ускорение работы программы

1. Заключение

По полученным данным видно, что время работы последовательной и параллельной программы увеличивается в зависимости от количества строк в файле. Как видно из рисунка, последовательный алгоритм в большинстве случаев отрабатывает быстрее чем параллельный. Наибольшего ускорения программа достигает, когда размер файла составляет 10 строк [3].

Параллельная программа работает медленнее чем последовательная, потому что имея разделяемые между потоками ресурсы мы жертвуем использованием локального кэша потока. Также достаточно много времени тратится на ожидание потоками друг друга [4].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Многопоточность в Java/Хабр. [Электронный ресурс] URL: https://habr.com/ru/post/164487/ (дата обращения: 26.10.2021).
2. Java Работа с файлами – Devcolibri. [Электронный ресурс] URL: <https://devcolibri.com/java-работа-с-файлами/> (дата обращения: 26.10.2021).
3. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления СПб.:БХВ-Петербург, 2002. - 608 с.
4. Грегори Р. Эндрюс Основы многопоточного, параллельного и распределенного программирования - Вильямс, 2002. - 512 с.

ПРИЛОЖЕНИЕ А

Код сигнального потока

import java.io.File;  
import java.io.IOException;  
import java.util.Scanner;  
import java.util.concurrent.locks.Condition;  
import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
  
class FileReaderThread extends Thread {  
  
 private final String filePathName;  
  
 private String currLine;  
 private boolean isMyTurn = false;  
 private boolean isMyWorkFinished = false;  
  
 private final Lock locker = new ReentrantLock();  
 private final Condition readingNextLine = locker.newCondition();  
 private final Condition readingNextLineIsFinished = locker.newCondition();  
  
  
 public FileReaderThread(String filePathName) {  
 this.filePathName = filePathName;  
 }  
  
 public void run() {  
  
 locker.lock();  
 try {  
  
 final Scanner scanner = new Scanner(new File(filePathName));  
 while (true) {  
 if (!isMyTurn) {  
 readingNextLine.await();  
 }  
  
 if (scanner.hasNextLine()) {  
 currLine = scanner.nextLine();  
 } else {  
 currLine = null;  
 }  
  
 isMyTurn = false;  
  
 if (isMyWorkFinished) {  
 scanner.close();  
 break;  
 }  
  
 readingNextLineIsFinished.signal();  
 }  
 } catch (IOException | InterruptedException e) {  
 e.printStackTrace();  
 } finally {  
 locker.unlock();  
 }  
 }  
  
 public void runReadLine() {  
 locker.lock();  
 isMyTurn = true;  
 readingNextLine.signal();  
 locker.unlock();  
 }  
  
 public String currLine() {  
 locker.lock();  
 if (isMyTurn) {  
 // causes the current thread to wait until it is signalled  
 readingNextLineIsFinished.awaitUninterruptibly();  
 }  
 locker.unlock();  
  
 return currLine;  
 }  
  
 public void finish() {  
 locker.lock();  
 isMyWorkFinished = true;  
 readingNextLine.signal();  
 locker.unlock();  
 }  
}

ПРИЛОЖЕНИЕ Б

Код класса мейн

import java.io.IOException;  
import java.io.PrintWriter;  
import java.nio.charset.StandardCharsets;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
  
public class Main {  
  
 public static final int *LINES\_NUM* = 10;  
 public static final int *SYMBOLS\_NUM* = 10;  
 public static boolean *SHOULD\_IT\_WORK\_IN\_PARALLEL* = false;  
 public static boolean *SHOULD\_IT\_DELETE\_FILES\_AFTER\_WORK* = true;  
  
 public static void main(String[] args) throws InterruptedException, IOException {  
  
 System.*out*.println("Лабораторная работа №.2---Выполнил студент группы 6132 Артамонов В.А.");  
 final String[] filesNames = *createFiles*(*LINES\_NUM*, *SYMBOLS\_NUM*);  
  
 if (*SHOULD\_IT\_WORK\_IN\_PARALLEL*) {  
 *parallel*(filesNames);  
 } else {  
 *sequential*(filesNames);  
 }  
 if (*SHOULD\_IT\_DELETE\_FILES\_AFTER\_WORK*) {  
 Files.*deleteIfExists*(Paths.*get*(filesNames[0]));  
 Files.*deleteIfExists*(Paths.*get*(filesNames[1]));  
 }  
 }  
  
 private static String[] createFiles(int linesNum, int symbolsNum) {  
  
 final String file1Name = "file1-" + linesNum + "x" + symbolsNum + ".txt";  
 final String file2Name = "file2-" + linesNum + "x" + symbolsNum + ".txt";  
  
 try (PrintWriter writer1 = new PrintWriter(file1Name, StandardCharsets.*UTF\_8*);  
 PrintWriter writer2 = new PrintWriter(file2Name, StandardCharsets.*UTF\_8*)) {  
  
 String strToWrite = "a".repeat(symbolsNum);  
  
 for (int rowIndex = 0; rowIndex < linesNum - 1; ++rowIndex) {  
 writer1.println(strToWrite);  
 writer2.println(strToWrite);  
 }  
  
 writer1.println("b");  
 writer2.println("c");  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 return new String[]{file1Name, file2Name};  
 }  
  
 private static void parallel(String[] filesNames) throws InterruptedException {  
 FileReaderThread potok1 = new FileReaderThread(filesNames[0]);  
 FileReaderThread potok2 = new FileReaderThread(filesNames[1]);  
 ParallelFilesComparatorThread parallel = new ParallelFilesComparatorThread(potok1, potok2);  
  
 potok1.start();  
 potok2.start();  
 parallel.start();  
   
 potok1.join();  
 potok2.join();  
 parallel.join();  
 }  
  
 private static void sequential(String[] filesNames) throws InterruptedException {  
 SequentialFilesComparatorThread posled = new SequentialFilesComparatorThread(filesNames);  
 posled.start();  
 posled.join();  
 }  
}

ПРИЛОЖЕНИЕ В

Код параллельного алгоритма

import java.util.Objects;  
  
public class ParallelFilesComparatorThread extends Thread {  
  
 private final FileReaderThread file1;  
 private final FileReaderThread file2;  
  
 public ParallelFilesComparatorThread(FileReaderThread file1, FileReaderThread file2) {  
 this.file1 = file1;  
 this.file2 = file2;  
 }  
  
 public void run() {  
  
 String file1Line;  
 String file2Line;  
 boolean isEndOfFiles;  
 int currStringsNum = 1;  
  
 long startMillis = System.*currentTimeMillis*();  
 while (true) {  
 file1.runReadLine();  
 file2.runReadLine();  
 file1Line = file1.currLine();  
 file2Line = file2.currLine();  
 isEndOfFiles = file1Line == null & file2Line == null;  
 if (isEndOfFiles) {  
 break;  
 }  
 if (!Objects.*equals*(file1Line, file2Line)) {  
 System.*out*.println("#" + currStringsNum + ":\n" +  
 "v faile 1 stroka = " + file1Line + "\n" +  
 "v faile 2 stroka = " + file2Line + "\n");  
 }  
 currStringsNum++;  
 }  
 long finishMillis = System.*currentTimeMillis*();  
 file1.finish();  
 file2.finish();  
 long consumedMillis = finishMillis - startMillis;  
 System.*out*.println("Consumed millis: " + consumedMillis);  
 }  
}

ПРИЛОЖЕНИЕ Г

Код последовательного алгоритма

import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.Objects;  
import java.util.Scanner;  
  
public class SequentialFilesComparatorThread extends Thread {  
  
 private final String[] filesNames;  
  
 public SequentialFilesComparatorThread(String[] filesNames) {  
 this.filesNames = filesNames;  
 }  
  
 public void run() {  
 try {  
 final Scanner file1Scanner = new Scanner(new File(filesNames[0]));  
 final Scanner file2Scanner = new Scanner(new File(filesNames[1]));  
  
 String file1Line;  
 String file2Line;  
 boolean isEndOfFiles;  
 int currStringsNum = 1;  
  
 long startMillis = System.*currentTimeMillis*();  
 while (true) {  
 if (file1Scanner.hasNextLine()) {  
 file1Line = file1Scanner.nextLine();  
 } else {  
 file1Line = null;  
 }  
  
 if (file2Scanner.hasNextLine()) {  
 file2Line = file2Scanner.nextLine();  
 } else {  
 file2Line = null;  
 }  
  
 isEndOfFiles = file1Line == null & file2Line == null;  
 if (isEndOfFiles) {  
 break;  
 }  
  
 if (!Objects.*equals*(file1Line, file2Line)) {  
 System.*out*.println("#" + currStringsNum + ":\n" +  
 "v faile 1 stroka = " + file1Line + "\n" +  
 "v faile 2 stroka = " + file2Line + "\n");  
 }  
  
 currStringsNum++;  
 }  
 long finishMillis = System.*currentTimeMillis*();  
  
 file1Scanner.close();  
 file2Scanner.close();  
  
 long consumedMillis = finishMillis - startMillis;  
 System.*out*.println("Consumed millis: " + consumedMillis);  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
}