Одесский национальный политехнический университет

Кафедра интеллектуальных систем и сетей

Лабораторная работа №7

По предмету «Системное программирование»

На тему «**Многопоточные приложения**»

Выполнили

Студенты группы АЕ-145

Михайлов О.М.

Зайцева М.О.

Проверил

Олещук О.В.

Одессы 2017

**Вариант задания:**

А. Создать многопоточное приложение с использованием Windows API, выполняющее перемножение матриц, размером 100\*100. Количество рабочих потоков равно (номер\_бригады+5). Исходные данные для работы всех потоков программы содержатся в одном или нескольких текстовых файлах (см. варианты заданий). Каждый рабочий поток считывает из файла очередные две матрицы и перемножает их. Результаты работы записываются в один или несколько выходных файлов (см. варианты заданий).

Б. Обеспечить корректный одновременный доступ к файлам со стороны множества потоков.

В. Обеспечить возможность корректной отмены работы указанного пользователем потока.

Г. Реализовать однопоточное приложение, выполняющего те же вычисления. Сравнить производительность одно- и многопоточного приложений.

Д. Отключить средства синхронизации потоков. Сравнить результаты работы программ с включенной и отключенной синхронизацией.

Примечание: для генерации файлов с исходными данными рекомендуется создать вспомогательную программу.

**Варианты заданий.**

1.

Средство синхронизации потоков: атомарная операция.

Тип приложения: консольное.

Файл с исходными данными: общий для всех потоков.

Файл с результатами работы: общий для всех потоков.

**Ход работы**

**package** laba.throw7;

**public class** CustomThread {

**public static void** main(String[] args) **throws** InterruptedException {

*//хрень по регистрации всех нужных переменных и буферов*

**int** size = 10;

**int**[][] matrix1;

**int**[][] matrix2;

**int**[][] result = **new int**[size][size];

Generator generator = **new** Generator(size);

Thread[] threads = **new** Thread[6];

*//создание матриц*

generator.generateMatrix();

matrix1 = generator.getMatrix();

matrix2 = generator.getMatrix();

*//создание потоков*

**for** (**int** i = 0; i < 6; i++) {

threads[i] = **new** Thread();

}

Multiplex multiplex = **new** Multiplex();

MyThreadMux myThreadMux = **new** MyThreadMux(multiplex, matrix1, matrix2);

**for** (Thread t: threads) {

t.start();

}

**for** (Thread t: threads) {

**try** {

t.join();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

*//результат*

myThreadMux.run();

result = multiplex.multiplexMatrix(matrix1, matrix2);

multiplex.printResult(result);

}

**private static class** MyThreadMux **implements** Runnable {

**private** Multiplex **multiplex**;

**private int**[][] **matrix1**;

**private int**[][] **matrix2**;

**public** MyThreadMux(Multiplex multiplex, **int**[][] matrix1, **int**[][] matrix2) {

**this**.**multiplex** = multiplex;

**this**.**matrix1** = matrix1;

**this**.**matrix2** = matrix2;

}

@Override

**public synchronized void** run() {

**multiplex**.multiplexMatrix(**matrix1**, **matrix2**);

}

}

}

**package** laba.throw7;

**public class** Multiplex {

**private int**[][] **result**;

**private int size**;

**public synchronized int**[][] multiplexMatrix(**int**[][] firstMatrix, **int**[][] secondMatrix) {

**size** = firstMatrix.**length**;

firstMatrix = firstMatrix;

secondMatrix = secondMatrix;

**result** = **new int**[**size**][**size**];

**int** sumLine = 0;

**for** (**int** i = 0; i < **size**; i++) {

**for** (**int** j = 0; j < **size**; j++) {

sumLine = getSum(i, j, firstMatrix, secondMatrix);

**result**[i][j] = sumLine;

}

}

**return result**;

}

**private int** getSum(**int** step1, **int** step2, **int**[][] firstMatrix, **int**[][] secondMatrix) {

**int** sum = 0;

**for** (**int** i = step1; i < **size**; i++) {

sum += firstMatrix[step1][i] \* secondMatrix[i][step2];

}

**return** sum;

}

**public void** printResult(**int**[][] matr) {

**for** (**int** i = 0; i < 10; i++) {

**for** (**int** j = 0; j < 10; j++) {

System.***out***.print(matr[i][j] + **" "**);

**if** (j == 9) {

System.***out***.println();

}

}

}

}

}

**package** laba.throw7;

**import** java.io.BufferedReader;

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileOutputStream;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.io.OutputStreamWriter;

**import** java.nio.charset.StandardCharsets;

**import** java.util.Arrays;

**public class** Generator {

**private int sizeMatrix**;

**public** Generator(**int** sizeMatrix) {

**this**.**sizeMatrix** = sizeMatrix;

}

**public void** generateMatrix() {

**int**[] matrix = **new int**[**sizeMatrix**\***sizeMatrix**];

**for** (**int** i = 0; i < **sizeMatrix**\***sizeMatrix**; i++) {

matrix[i] = 0 + (**int**) (Math.*random*() \* 10);

}

File file = **new** File(**"matrix.txt"**);

BufferedWriter bufferedWriter = **null**;

**try** {

bufferedWriter = **new** BufferedWriter(

**new** OutputStreamWriter(**new** FileOutputStream(file),

StandardCharsets.***UTF\_8***));

bufferedWriter.write(Arrays.*toString*(matrix));

bufferedWriter.flush();

bufferedWriter.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

**public int**[][] getMatrix() {

String line = **null**;

**int**[][] matrix = **new int**[**sizeMatrix**][**sizeMatrix**];

**try** {

BufferedReader bufferedReader2 = **new** BufferedReader(

**new** FileReader(**"/home/mikha/Документы/Universe/Labs/matrix.txt"**));

line = bufferedReader2.readLine();

} **catch** (IOException e) {

e.printStackTrace();

}

**assert** line != **null**;

line = line.substring(1, line.length() - 1);

**int** size = 0;

**int** pointer1 = 0;

**int** pointer2 = 0;

**char**[] tmp = line.toCharArray();

**while** (size < line.length()) {

**if** (tmp[size] > 47 && tmp[size] < 58) {

**if** (pointer1 <= **sizeMatrix**) {

matrix[pointer1][pointer2] = (**int**) tmp[size] - **'0'**;

**if** (pointer2 == **sizeMatrix**-1) {

pointer1++;

pointer2 = 0;

}**else** {

pointer2++;

}

}

}

size++;

}

**return** matrix;

}

}

**Вывод:** В ходе данной лабораторной работы была разработана консольная программа, которая выполняется на 6 потоках одновременно и выводит нам вариант умножения двух матриц 100х100. Многопоточность обеспечивает возможность оптимизировать и ускорить обработку разных, в данном случае, независимых друг от друга задач.