Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**Тема работы**

Студент: Фаттяхетдинов Сильвестр Динарович

Группа: М8О-208Б-20

Вариант: 11

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/silverfatt/OS/tree/main/os\_lab3

**Постановка задачи**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков должно быть задано ключом запуска вашей программы. Наложить K раз фильтры эрозии и наращивания на матрицу, состоящую из вещественных чисел. На выходе получается 2 результирующие матрицы.

**Общие сведения о программе**

Вся программа содержится в одном файле main.cpp

**Общий метод и алгоритм решения**

Запуск осуществляется при помощи ввода в командную строку unix:

./main

Считываются следующие данные: количество применения фильтра (К), количество потоков (N), матрица эрозии, матрица наращивания и матрица изображения. Для упрощения передачи в потоки все матрицы хранятся в динамически выделенных одномерных массивах.  
Далее запускается некоторое количество потоков (не больше N), в которые передаётся функция prep. Распараллеливание происходит по следующему правилу – каждый новый поток обрабатывает новую строку. Пример для матрицы свёртки размером 3:  
Матрица изображения:  
1 2 3 4  
5 6 7 8  
9 10 11 12  
13 14 15 16

Первый поток обработает:

1 2 3 2 3 4

5 6 7 6 7 8

9 10 11 10 11 12

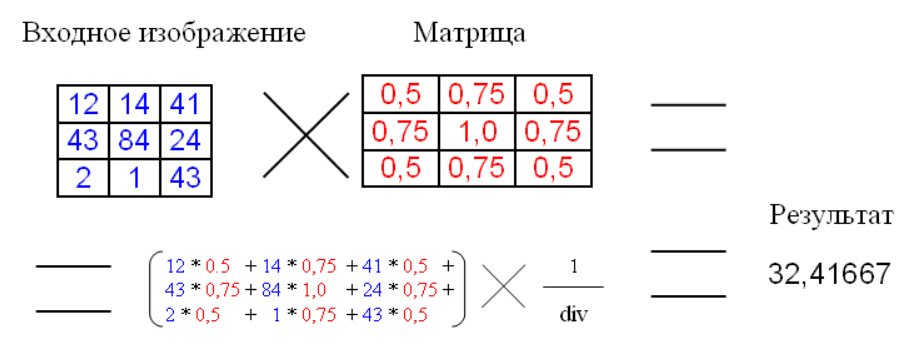
Второй поток обработает:

5 6 7 6 7 8

9 10 11 10 11 12

13 14 15 14 15 16

Prep – функция, применяющая матрицу свёртки – принимает следующие аргументы – номер обрабатываемой строки, размеры матрицы изображения, размер матрицы свёртки и сами матрицы, а также массив результатов (общий для всех потоков). Происходит конвертация в двумерные массивы для упрощения работы с матрицами. Затем выполняется преобразование согласно следующему правилу:



Div в данном случае принимается за 1. Результаты всех потоков записываются в общий для них массив results. После завершения работы всех процессов краевые элементы матрицы изображения заменяются на 0, а все остальные – числами из results. Данный процесс повторяется для обеих матриц свёртки K раз.

**Исходный код**

main.cpp

#include <iostream>

#include <fstream>

#include <thread>

#include <vector>

#include <unistd.h>

using namespace std;

void dosmth(){

    //cout << "a" << endl;

    sleep(10);

}

void prep(int str\_num, int n, int m, int s, double \*matrix, double \*sviortka, double\* results){

    double mmatrix[n][m];

    double msviortka[s][s];

    double mresults[n-2];

    int q = 0;

    for (int i = 0; i<n; i++){

        for (int j = 0; j<m; j++){

            mmatrix[i][j] = matrix[q];

            q++;

        }

    }

    q = 0;

    for (int i = 0; i<s; i++){

        for (int j = 0; j<s; j++){

            msviortka[i][j] = sviortka[q];

            q++;

        }

    }

    for(int i = 0; i < m - s + 1; i++){

        double res = 0;

        int j\_core = 0, k\_core = 0;

        for(int j = str\_num; j < str\_num + s; j++){

            for(int k = i; k < i + s; k++){

                //cout << "Elements:" <<  mmatrix[j][k] << " " << msviortka[j\_core][k\_core] << " " << j\_core << " " << k\_core <<endl;

                //cout << mmatrix[j][k] \* msviortka[j\_core][k\_core] << " " << res << endl;

                res += mmatrix[j][k] \* msviortka[j\_core][k\_core];

                k\_core++;

            }

            k\_core = 0;

            j\_core++;

        }

        mresults[i] = res;

        // for(int i = 0; i < n - 2; i++){

        //     cout << mresults[i] << " ";

        // }

        // cout << "Result:" << res << endl;

    }

    for (int i = 0; i < n-2; i++){

        results[str\_num \* (n - s + 1) + i] = mresults[i];

    }

}

int main(){

    int n\_errosion;

    int K, N;

    cout << "Insert K: " << endl;

    cin >> K;

    cout << "Insert number of threads. Insert -1 if there are no restrictions: " << endl;

    cin >> N;

    if(!N || (N < 0 && N != -1)){

        cout << "Error, incorrect amount of threads" << endl;

        return 1;

    }

    cout << "Insert size of errosion matrix: " << endl;

    cin >> n\_errosion;

    double \*errosion = new double[n\_errosion\*n\_errosion];

    cout << "Insert errosion matrix elements: " << endl;

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

            cin >> errosion[i];

    }

    //blinking matrix

    cout << "Insert size of blinking matrix: ";

    int n\_blinking;

    cin >> n\_blinking;

    double \*blinking = new double[n\_blinking\*n\_blinking];

    cout << "Insert blinking matrix elements: " << endl;

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

        cin >> blinking[i];

    }

    //matrix of image

    cout << "Insert amount of strings and columns of image matrix: " << endl;

    int n, m;

    cin >> n >> m;

    double \*matrix = new double[n\*m];

    double \*matrix2 = new double[n\*m];

    cout << "Insert image matrix elements: " << endl;

    for(int i = 0; i < n\*m; i++){

        cin >> matrix[i];

        matrix2[i] = matrix[i];

    }

    thread th\_e[n-n\_errosion+1];

    thread th\_b[n-n\_blinking+1];

    double \*e\_results = new double[(n-2)\*(m-2)];

    for(int i = 0; i < (n - 2) \* (m - 2); i++){

        e\_results[i] = 0;

    }

    double \*b\_results = new double[(n-2)\*(m-2)];

    for(int i = 0; i < (n - 2) \* (m - 2); i++){

        b\_results[i] = 0;

    }

    while(K > 0){

        if(N == -1 || N >= n - n\_errosion + 1){

            for(int i = 0; i < n - n\_errosion + 1; i++){

                th\_e[i] = thread(prep, i, n, m, n\_errosion, matrix, errosion, e\_results);

            }

            for(int i = 0; i < n - n\_errosion + 1; i++){

                th\_e[i].join();

            }

        }

        else {

            for(int i = 0; i < n - n\_errosion + 1; i++){

                if(i >= N && i != 0){

                    th\_e[i - N].join();

                }

                th\_e[i] = thread(prep, i, n, m, n\_errosion, matrix, errosion, e\_results);

            }

            for(int i = n - n\_errosion + 1 - N; i < n - n\_errosion + 1; i++){

                th\_e[i].join();

            }

        }

        int q = 0;

        for (int j = 0; j < n \* m; j++){

            if ((j >= 0 && j < m) || (j % m == 0) || ((j + 1) % m == 0) || (j < n \* m && j > n \* m - m)){

                matrix[j] = 0;

            } else {

                matrix[j] = e\_results[q];

                q++;

            }

        }

        if(N == -1 || N >= n - n\_blinking + 1){

            for(int i = 0; i < n - n\_blinking + 1; i++){

                th\_b[i] = thread(prep, i, n, m, n\_blinking, matrix2, blinking, b\_results);

            }

            for(int i = 0; i < n - n\_blinking + 1; i++){

                th\_b[i].join();

            }

        }

        else {

            for(int i = 0; i < n - n\_blinking + 1; i++){

                if(i >= N && i != 0){

                    th\_b[i - N].join();

                }

                th\_b[i] = thread(prep, i, n, m, n\_blinking, matrix2, blinking, b\_results);

            }

            for(int i = n - n\_blinking + 1 - N; i < n - n\_blinking + 1; i++){

                th\_b[i].join();

            }

        }

        int p = 0;

        for (int j = 0; j < n \* m; j++){

            if ((j >= 0 && j < m) || (j % m == 0) || ((j + 1) % m == 0) || (j < n \* m && j > n \* m - m)){

                matrix2[j] = 0;

            } else {

                matrix2[j] = b\_results[p];

                p++;

            }

        }

        K--;

    }

    cout << "Erosion result: " << endl;

    for (int i = 0; i<(n\*m); i++){

        cout << matrix[i] << " ";

        if ((i+1) % (m) == 0) cout << endl;

    }

    cout << "Blinking result: " << endl;

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

        cout << matrix2[i] << " ";

        if ((i+1) % (m) == 0) cout << endl;

    }

    delete[] matrix;

    delete[] matrix2;

    delete[] errosion;

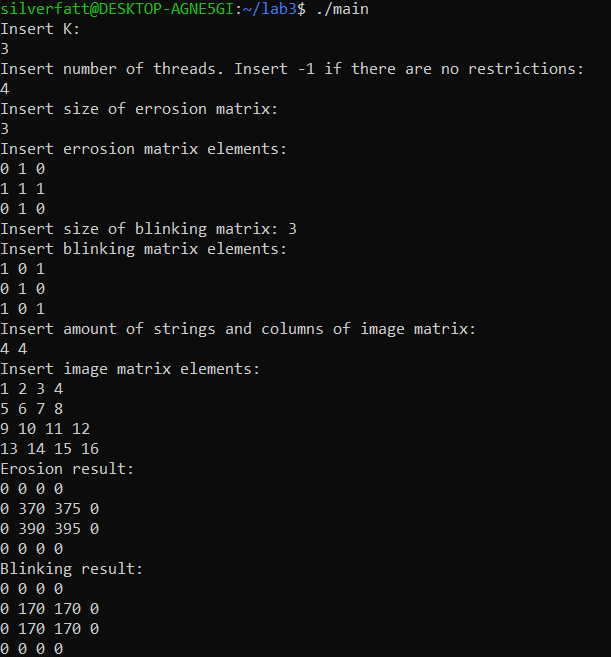
    delete[] blinking;

    delete[] e\_results;

    delete[] b\_results;

}

**Демонстрация работы программы**



**Выводы**

Я приобрёл навыки в управлении потоками в ОС Unix и распараллеливании выполнения сложных задач для увеличения производительности.