Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа № 3 по курсу

«Операционные системы»

Студент: Постнов Александр Вячеславович
Группа: М8О-201Б-21
Вариант: 11
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

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/kappaprideonly/mai-os-labs

Постановка задачи

Наложить K раз фильтры эрозии и наращивания на матрицу, состоящую из вещественных чисел. На выходе получается 2 результирующие матрицы

Общие сведения о программе

CMakeLists.txt - описание процесса сборки проекта

main.cpp - считывание и вывод данных

lab3.h - заголовочный файл, описаны функции для работы с потоками и использование фильтров и наращивания

lab3.cpp - реализация функций, которые определены в lab3.h

utils.h - полезные функции

lab3_test.cpp - тесты для программы, реализованные с помощью gtest

Общий метод и алгоритм решения

```
Реализовал функции наращивания и эрозии для матрицы для одной клетки.
Распределил все клетки матрицы между потоками.
```

Исходный код

CMakeLists.txt

```
add_executable(lab3 main.cpp include/lab3.h src/lab3.cpp include/utils.h)
```

```
target_include_directories(lab3 PRIVATE include)

target_link_libraries(lab3 PRIVATE Threads::Threads)
```

lab3.h

```
#ifndef OS_LABS_LAB3_H
#define OS_LABS_LAB3_H

#include <vector>
#include <iostream>

using TMatrix = std::vector<std::vector<float>>;

struct TThreadToken {
   std::vector <std::pair<int, int>> coords;
   int counter;
   TMatrix* matrix;
   TMatrix* filter;
   TMatrix* resultMatrix;
};

void CheckingAround(int row, int col, TMatrix &matrix, TMatrix &filter,
TMatrix &resultMatrix);

void SummingAround(int row, int col, TMatrix &matrix, TMatrix &filter,
TMatrix &resultMatrix, int counter);

void ReadMatrix(TMatrix &matrix);

void WriteMatrix(TMatrix &matrix);

void* DilationRoutine(void* arg);

void* ErosionRoutine(void* arg);

void DilationMatrix(TMatrix &matrix, TMatrix &filter, TMatrix
&resultDilation, int threadCount, int counter);

void ErosionMatrix(TMatrix &matrix, TMatrix &filter, TMatrix
&resultErosion, int threadCount, int counter);

#endif //OS_LABS_LAB3_HOS_LABS_LAB3_H
```

lab3.cpp

```
#include "lab3.h"
#include "utils.h"

#include <pthread.h>


pthread_mutex_t mutex;

void CheckingAround(int row, int col, TMatrix &matrix, TMatrix &filter,
TMatrix &resultMatrix){
    // координаты для проверки, row и col "приделываем к центру filter"
    int rowBegin = row - Isize(filter) / 2;
    int colBegin = col - Isize(filter[0]) / 2;
    int flag = 1;
    for (int i = 0; i < Isize(filter); i++) {
        if (flag == 0) {
            break;
        }
        for (int j = 0; j < Isize(filter[i]); j++) {
            int rowTemp = rowBegin + i;
            int colTemp = colBegin + j;
            if (!(rowTemp >= 0 && rowTemp < Isize(matrix) && colTemp >= 0
&& colTemp < Isize(matrix[0]) && filter[i][j] ==
matrix[rowTemp][colTemp])) {
                flag = 0;
                break;
            }
        }
    }
    if (!flag) {
        resultMatrix[row][col] = 0;
    }
}

void SummingAround(int row, int col, TMatrix &matrix, TMatrix &filter,
TMatrix &resultMatrix, int counter) {
    // координаты для суммирования, row и col "приделываем к центру filter"
    int rowBegin = row - Isize(filter) / 2;
    int colBegin = col - Isize(filter[0]) / 2;
    for (int i = 0; i < Isize(filter); i++) {
        for (int j = 0; j < Isize(filter[i]); j++) {
            int rowTemp = rowBegin + i;
            int colTemp = colBegin + j;
```

```
            if (rowTemp >= 0 && rowTemp < Isize(matrix) && colTemp >= 0 &&
colTemp < Isize(matrix[0])) {
                pthread_mutex_lock(&mutex);
                resultMatrix[rowTemp][colTemp] += filter[i][j] *
(float)counter;
                pthread_mutex_unlock(&mutex);
            }
        }
    }
}

void WriteMatrix(TMatrix &matrix) {
    for (int i = 0; i < Isize(matrix); i++) {
        for (int j = 0; j < Isize(matrix[i]); j++) {
            std::cout << matrix[i][j] << " ";
        }
        std::cout << "\n";
    }
}

void ReadMatrix(TMatrix &matrix) {
    for (int i = 0; i < Isize(matrix); i++) {
        for (int j = 0; j < Isize(matrix[i]); j++) {
            std::cin >> matrix[i][j];
        }
    }
}

void* DilationRoutine(void* arg) {
    auto* token = (TThreadToken*) arg;
    for (int i = 0; i < Isize(token->coords); i++) {
        SummingAround(token->coords[i].first, token->coords[i].second,
*token->matrix, *token->filter, *token->resultMatrix, token->counter);
    }
    return nullptr;
}

void* ErosionRoutine(void* arg) {
    auto* token = (TThreadToken*) arg;
    for (int i = 0; i < Isize(token->coords); i++) {
        CheckingAround(token->coords[i].first, token->coords[i].second,
*token->matrix, *token->filter, *token->resultMatrix);
    }
    return nullptr;
}
```

```
void DilationMatrix(TMatrix &matrix, TMatrix &filter, TMatrix
&resultDilation, int threadCount, int counter) {
    TMatrix matrixCopy = matrix;
    pthread_mutex_init(&mutex, nullptr);
    std::vector <pthread_t> threads(threadCount);
    std::vector <TThreadToken> tokens(threadCount);
    // заполнение информации для токенов
    for (int i = 0; i < threadCount; i++) {
        tokens[i].matrix = &matrixCopy;
        tokens[i].filter = &filter;
        tokens[i].resultMatrix = &resultDilation;
        tokens[i].counter = counter;
    }
    int update = 0;
    for (int j = 0; j < Isize(matrix[0]); j++) {
        for (int i = 0; i < Isize(matrix); i++) {
            update++;
            tokens[update % threadCount].coords.emplace_back(i, j);
        }
    }

    for (int i = 0; i < threadCount; i++) {
        pthread_create(&threads[i], nullptr, &DilationRoutine, &tokens[i]);
    }
    for (int i = 0; i < threadCount; i++) {
        pthread_join(threads[i], nullptr);
    }

    pthread_mutex_destroy(&mutex);
}


void ErosionMatrix(TMatrix &matrix, TMatrix &filter, TMatrix
&resultErosion, int threadCount, int counter) {
    TMatrix matrixCopy = matrix;
    std::vector <pthread_t> threads(threadCount);
    std::vector <TThreadToken> tokens(threadCount);
    // заполнение информации для токенов
    for (int i = 0; i < threadCount; i++) {
        tokens[i].matrix = &matrixCopy;
        tokens[i].filter = &filter;
        tokens[i].resultMatrix = &resultErosion;
        tokens[i].counter = counter;
```

```
    }
    int update = 0;
    for (int i = 0; i < Isize(matrix); i++) {
        for (int j = 0; j < Isize(matrix[i]); j++) {
            update++;
            tokens[update % threadCount].coords.emplace_back(i, j);
        }
    }

    for (int k = 0; k < counter; k++) {
        for (int i = 0; i < threadCount; i++) {
            pthread_create(&threads[i], nullptr, &ErosionRoutine,
&tokens[i]);
        }
        for (int i = 0; i < threadCount; i++) {
            pthread_join(threads[i], nullptr);
        }
        matrixCopy = resultErosion;
    }
}
```

utils.h

```
#ifndef OS_LABS_UTILS_H
#define OS_LABS_UTILS_H

template <typename Container>
inline int Isize(const Container& cont) {
    return static_cast<int>(cont.size());
}

#endif //OS_LABS_UTILS_H
```

main.cpp

```
#include "lab3.h"


int main() {
    int threadCount;
    int rowMatrix;
    int colMatrix;
```

```
    int rowFilter;
    int colFilter;
    int counter;

    std::cin >> threadCount;

    std::cin >> rowMatrix >> colMatrix;
    TMatrix matrix(rowMatrix, std::vector <float>(colMatrix));
    ReadMatrix(matrix);

    std::cin >> rowFilter >> colFilter;
    TMatrix filter(rowFilter, std::vector<float>(colFilter));
    ReadMatrix(filter);

    std::cin >> counter;

    TMatrix resultDilation = matrix;
    TMatrix resultErosion = matrix;

    DilationMatrix(matrix, filter, resultDilation, threadCount, counter);
    ErosionMatrix(matrix, filter, resultErosion, threadCount, counter);

    WriteMatrix(resultDilation);
    WriteMatrix(resultErosion);
}
```

lab3_test.cpp

```
#include <cstdlib>
#include <gtest/gtest.h>

#include <lab3.h>
#include <utils.h>

#include <chrono>


namespace {
    TMatrix GenerateMatrix(int n, int m) {
        TMatrix result(n, std::vector<float>(m));
```

```
        std::srand(std::time(nullptr));

        for(int i = 0; i < n; ++i) {
            for(int j = 0; j < m; ++j) {
                result[i][j] = (float)(std::rand() % 100);
            }
        }

        return result;
    }
}


bool operator==(const TMatrix& lhs, const TMatrix& rhs) {
    if(lhs.size() != rhs.size()) {
        return false;
    }

    for(int i = 0; i < Isize(lhs); ++i) {
        if(lhs[i].size() != rhs[i].size()) {
            return false;
        }

        for(int j = 0; j < Isize(lhs); ++j) {
            if(lhs[i][j] != rhs[i][j]) {
                return false;
            }
        }
    }

    return true;
}



TEST(Lab3Test, CheckingAroundTest) {
    TMatrix matrix = {
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5}
    };
```

```
    TMatrix filter = {
        {1, 2, 3},
        {1, 2, 3},
        {1, 2, 3}
    };

    TMatrix resultMatrix = matrix; // изначально копия

    std::vector <std::pair<int, int>> checkedCoords = {
        {2, 0},
        {3, 0},
        {2, 2},
        {3, 4},
        {2, 4},
        {1, 1},
        {2, 1}
    };

    for (int i = 0; i < Isize(checkedCoords); i++) {
        int row = checkedCoords[i].first;
        int col = checkedCoords[i].second;
        CheckingAround(row, col, matrix, filter, resultMatrix);
    }

    TMatrix expectedMatrix = {
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5},
        {0, 2, 0, 4, 0},
        {0, 2, 3, 4, 0},
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5}
    };

    EXPECT_EQ(resultMatrix, expectedMatrix);
}

TEST(Lab3Test, SummingAroundTest) {
    TMatrix matrix = {
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5}
```

```
    };

    TMatrix filter = {
        {1, 2, 3},
        {1, 2, 3},
        {1, 2, 3}
    };

    TMatrix resultMatrix = matrix; // изначально копия

    std::vector <std::pair<int, int>> summingCoords = {
        {1, 1},
        {2, 1},
        {5, 4}
    };

    for (int i = 0; i < Isize(summingCoords); i++) {
        int row = summingCoords[i].first;
        int col = summingCoords[i].second;
        SummingAround(row, col, matrix, filter, resultMatrix, 1);
    }

    TMatrix expectedMatrix = {
        {2, 4, 6, 4, 5},
        {3, 6, 9, 4, 5},
        {3, 6, 9, 4, 5},
        {2, 4, 6, 4, 5},
        {1, 2, 3, 5, 7},
        {1, 2, 3, 5, 7}
    };

    EXPECT_EQ(resultMatrix, expectedMatrix);
}


TEST(Lab3Test, SingleThreadYieldsCorrectResults) {
    int countTests = 3;
    std::vector <TMatrix> expectedMatrixsErosion {
        {
            {1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1}
        },
```

```
        {
            {0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0}
        },
        {
            {0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 1, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0},
        }
    };

    std::vector <TMatrix> expectedMatrixsDilation {
        {
            {2, 2, 2, 2, 2},
            {2, 2, 2, 2, 2},
            {2, 2, 2, 2, 2},
            {2, 2, 2, 2, 2},
            {2, 2, 2, 2, 2}
        },
        {
            {13, 19, 19, 19, 13},
            {19, 28, 28, 28, 19},
            {19, 28, 28, 28, 19},
            {19, 28, 28, 28, 19},
            {13, 19, 19, 19, 13}
        },
        {
            {9, 13, 13, 13, 9},
            {13, 19, 19, 19, 13},
            {13, 19, 19, 19, 13},
            {13, 19, 19, 19, 13},
            {9, 13, 13, 13, 9}
        }
    };


    std::vector <TMatrix> matrixs {
        {
            {1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1},
```

```
            {1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1},
        },
        {
            {1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1},
        },
        {
            {1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1},
            {1, 1, 1, 1, 1},
        }
    };

    std::vector <TMatrix> filters {
        {
            {1}
        },
        {
            {3, 3, 3},
            {3, 3, 3},
            {3, 3, 3},
        },
        {
            {1, 1, 1},
            {1, 1, 1},
            {1, 1, 1}
        }

    };
    std::vector <int> counters {
        1,
        1,
        2
    };

    for (int i = 0; i < countTests; i++) { // for erosion
        TMatrix expectedMatrix = expectedMatrixsErosion[i];
        TMatrix matrix = matrixs[i];
```

```
        TMatrix filter = filters[i];
        int counter = counters[i];
        TMatrix resultErosion = matrix;
        ErosionMatrix(matrix, filter, resultErosion, 1, counter);
        EXPECT_EQ(resultErosion, expectedMatrix);
    }

    for (int i = 0; i < countTests; i++) { // for dillation
        TMatrix expectedMatrix = expectedMatrixsDilation[i];
        TMatrix matrix = matrixs[i];
        TMatrix filter = filters[i];
        int counter = counters[i];
        TMatrix resultDilation = matrix;
        DilationMatrix(matrix, filter, resultDilation, 1, counter);
        EXPECT_EQ(resultDilation, expectedMatrix);
    }
}

TEST(Lab3Test, ThreadConfigurations) {
    std::srand(std::time(nullptr));
    auto performTestForGivenSize = [](int n1, int m1, int n2, int m2, int
maxThreadCount) {
        int counter = 1 + std::rand() % 8;

        auto matrix = GenerateMatrix(n1, m1);
        auto filter = GenerateMatrix(n2, m2);

        auto resultDilationOne = matrix;
        auto resultErosionOne = matrix;

        DilationMatrix(matrix, filter, resultDilationOne, 1, counter);
        ErosionMatrix(matrix, filter, resultErosionOne, 1, counter);

        for(int i = 2; i < maxThreadCount; ++i) {
            auto resultDilation = matrix;
            auto resultErosion = matrix;
            DilationMatrix(matrix, filter, resultDilation, i, counter);
            ErosionMatrix(matrix, filter, resultErosion, i, counter);
            EXPECT_EQ(resultDilation, resultDilationOne);
            EXPECT_EQ(resultErosion, resultErosionOne);
        }
    };

    performTestForGivenSize(3, 3, 1, 1, 2);
    performTestForGivenSize(10, 10, 3, 3, 2);
```

```
    performTestForGivenSize(100, 100, 7, 7, 12);
}


TEST(Lab3Test, PerfomanceTest) {
    auto getAvgTime = [](int threadCount) {
        auto matrix = GenerateMatrix(1000, 1000);
        auto filter = GenerateMatrix(5, 5);

        constexpr int runsCount = 3;
        constexpr int counter = 5;

        double avg = 0;

        for(int i = 0; i < runsCount; ++i) {
            auto begin = std::chrono::high_resolution_clock::now();
            auto resultErosion = matrix;
            auto resultDilation = matrix;
            ErosionMatrix(matrix, filter, resultErosion, threadCount,
counter);
            auto end = std::chrono::high_resolution_clock::now();
            avg +=
std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin).count();
        }

        return avg / runsCount;
    };

    auto singleThread = getAvgTime(1);
    auto multiThread = getAvgTime(8);

    std::cout << "Avg time for 1 thread: " << singleThread << '\n';
    std::cout << "Avg time for 8 threads: " << multiThread << '\n';

    EXPECT_GE(singleThread, multiThread);
}
```

Демонстрация работы программы

Start 2: lab3_test

2: Test command: /home/alex/mai-os-labs/build/tests/lab3_test

2: Working Directory: /home/alex/mai-os-labs/build/tests

2: Test timeout computed to be: 10000000

2: Running main() from /home/alex/mai-os-labs/build/_deps/googletest-src/googletest/src/gtest_main.cc

2: [==========] Running 5 tests from 1 test suite.

2: [----------] Global test environment set-up.

2: [----------] 5 tests from Lab3Test

2: [RUN] Lab3Test.CheckingAroundTest

2: [OK] Lab3Test.CheckingAroundTest (0 ms)

2: [RUN] Lab3Test.SummingAroundTest

2: [OK] Lab3Test.SummingAroundTest (0 ms)

2: [RUN] Lab3Test.SingleThreadYieldsCorrectResults

2: [OK] Lab3Test.SingleThreadYieldsCorrectResults (0 ms)

2: [RUN] Lab3Test.ThreadConfigurations

2: [OK] Lab3Test.ThreadConfigurations (556 ms)

2: [RUN] Lab3Test.PerfomanceTest

2: Avg time for 1 thread: 370

2: Avg time for 8 threads: 150

2: [OK] Lab3Test.PerfomanceTest (1638 ms)

2: [----------] 5 tests from Lab3Test (2196 ms total)

2:

2: [----------] Global test environment tear-down

2: [==========] 5 tests from 1 test suite ran. (2196 ms total)

2: [PASSED] 5 tests.

2/5 Test #2: lab3_test .....   Passed   2.20 sec

test 3

Выводы

Я приобрел практические навыки в:

1) Управление потоками в ОС
2) Обеспечение синхронизации между потоками