Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу
«Операционные системы»

Многопоточные программы

Студент: Калиниченко Артём Андреевич
Группа: М8О–210Б–22
Вариант: 10
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023.

Постановка задачи

Цель работы
Целью является приобретение практических навыков в:

- Написании многопоточных программ.
- Ознакомлении с библиотеками для написания многопоточных программ

Задание
Требуется решить систему линейных уравнений методом Гаусса

Общие сведения о программе

Программа компилируется из файла main.cpp. Также используется заголовочные файлы: iostream, vector, thread, fstream, random.

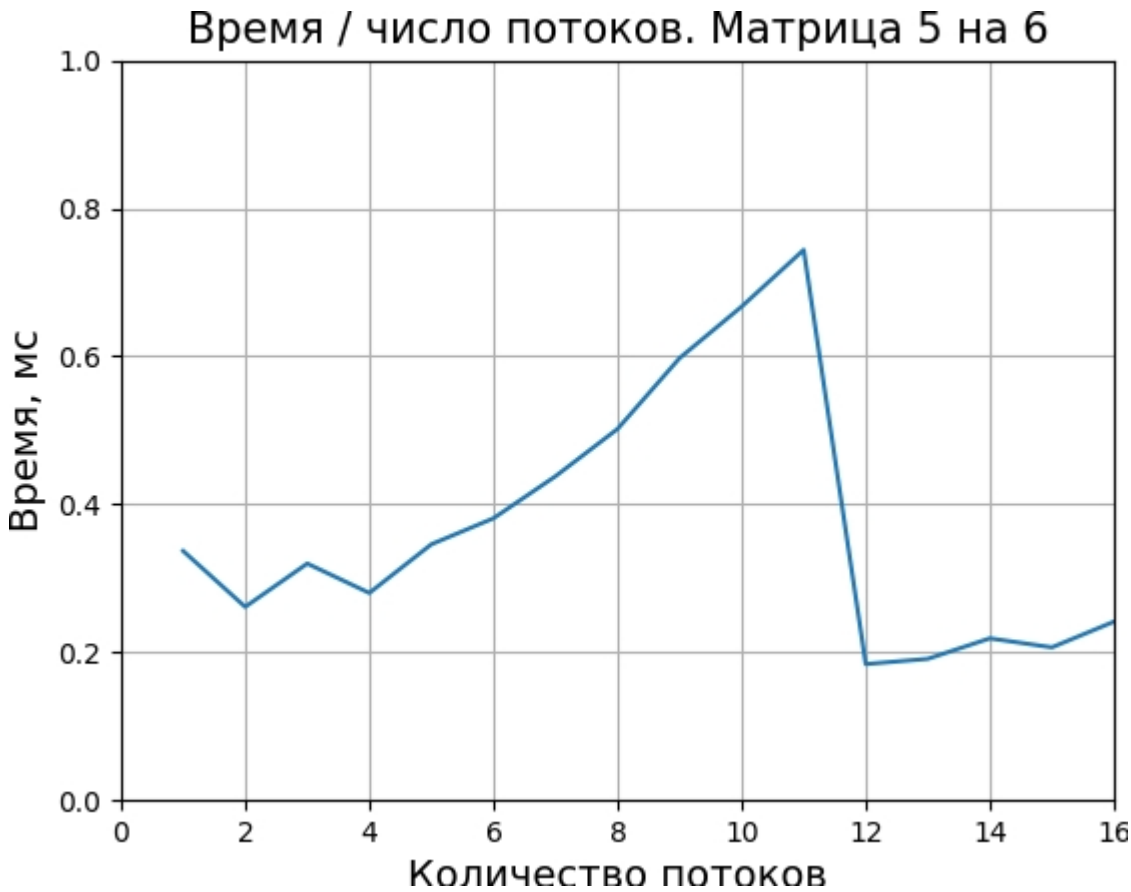
Общий метод и алгоритм решения.

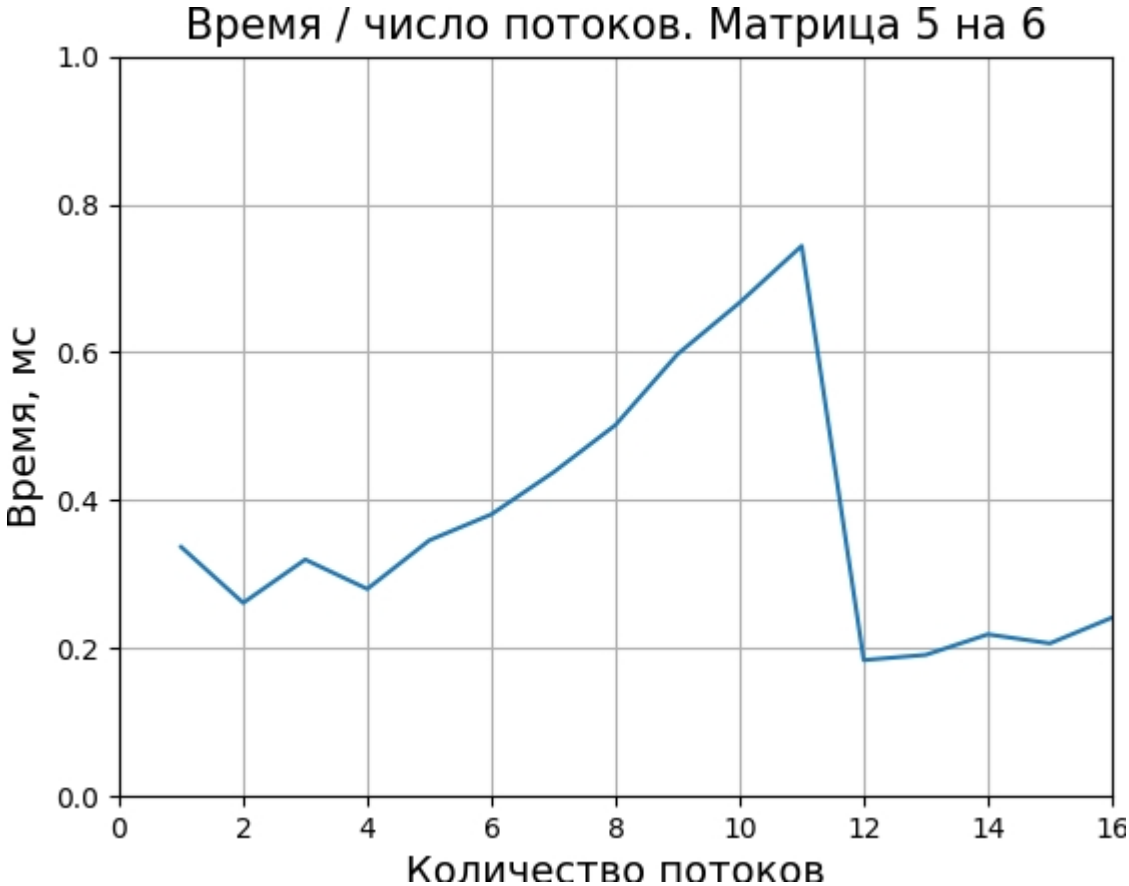
Для реализации поставленной задачи необходимо:

1. Изучить принципы работы работы многопоточных программ.
2. Написать функцию, осуществляющую приведение к треугольному виду части матрицы и функцию реализации метода Гаусса с использованием потоков.

Исследование скорости выполнения программы.

После реализации программы необходимо было проверить, с какой скоростью работают написанные алгоритмы. Для этого мной был написан скрипт, который записывает в текстовый файл время работы каждого из алгоритмов. После этого была составлена таблица, по которой был построен график.


Время / число потоков. Матрица 5 на 6
Время, мс
Количество потоков
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
14
16


Время / число потоков. Матрица 5 на 6
Время, мс
Количество потоков
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
12
14
16

Было замечено, что при небольшом размере матрицы по времени выигрывает однопоточная реализация, потому что больше ресурсов затрачисвалось на создание потоков. Однако, с ростом размера матрицы, время выполнения программы убывало с ростом числа потоков.

Основные файлы программы

vis.py:

```
import matplotlib.pyplot as plt
import csv

X = []
Y = []

with open('data.csv', 'r') as datafile:
```

```
    plotting = csv.reader(datafile, delimiter=';')

    for ROWS in plotting:
        X.append(float(ROWS[0]))
        Y.append(float(ROWS[1]))

plt.plot(X, Y)
plt.xlim([0, 16])
plt.ylim([0, 3500])
plt.ylabel(r'Время, мс', fontsize = 14)
plt.xlabel(r'Количество потоков', fontsize = 14)
plt.title(r'Время / число потоков. Матрица 1000 на 1001', fontsize = 15)
plt.grid(True)
plt.show()
```

main.cpp

```
#include <iostream>
#include <vector>
#include <thread>
#include <fstream>
#include <random>

using namespace std;

//Функция вывода матрицы
void PrintMatrix(vector<vector<double>> matrix) {
    int rows = matrix.size();
    int cols = matrix[0].size();

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

//Функция реализации метода Гаусса на части матрицы
void GaussEliminationPart(vector<vector<double>>& matrix, int start_row, int
end_row) {
```

```
    int cols = matrix[0].size();

    for (int i = start_row; i < end_row; i++) {
        double max_elem = matrix[i][i];
        int max_row = i;

        for (int j = i + 1; j < matrix.size(); j++) {
            if (abs(matrix[j][i]) > abs(max_elem)) {
                max_elem = matrix[j][i];
                max_row = j;
            }
        }

        if (max_row > i) {
            swap(matrix[i], matrix[max_row]);
        }

        for (int j = i + 1; j < matrix.size(); j++) {
            double factor = matrix[j][i] / matrix[i][i];
            for (int k = i; k < cols; k++) {
                matrix[j][k] -= factor * matrix[i][k];
            }
        }
    }
}

// Функция реализации метода Гаусса с использованием потоков
void GaussEliminationThreaded(vector<vector<double>> &matrix, int
num_threads) {
    int rows = matrix.size();
    int part_size = rows / num_threads; //размер части матрицы, для каждого
потока

    vector<thread> threads;
    int start_row = 0;
    int end_row = 0;

    for (int i = 0; i < num_threads - 1; i++) {
        end_row = start_row + part_size;
```

```
        threads.push_back(thread(GaussEliminationPart, ref(matrix), start_row,
end_row));
        start_row = end_row;
    }

    end_row = rows;
    threads.push_back(thread(GaussEliminationPart, ref(matrix), start_row,
end_row));

    for (int i = 0; i < num_threads; i++) {
        threads[i].join(); //метод join() для каждого потока, чтобы дождаться их
завершения
    }
}

// Функция обратной замены
vector<double> backSubstitution(vector<vector<double>> matrix) {
    int rows = matrix.size();
    int cols = matrix[0].size();

    vector<double> solution(rows);
    for (int i = rows - 1; i >= 0; i--) {
        solution[i] = matrix[i][cols - 1];
        for (int j = i + 1; j < rows; j++) {
            solution[i] -= matrix[i][j] * solution[j];
        }
        solution[i] /= matrix[i][i];
    }

    return solution;
}


// Функция решения СЛАУ с использованием потоков
vector<vector<double>> SolveSystemThreaded(vector<vector<double>> matrix,
int num_threads) {
    GaussEliminationThreaded(matrix, num_threads);
    return matrix;
}
```

```
int main() {
    int n, m;
    cout << "Enter size of matrix" << endl;
    cin >> n >> m;

    int num_threads;
    cout << "Enter number of threads (from 1 to threads)" << endl;
    cin >> num_threads;

    vector<vector<double>> matrix(n, vector<double>(m));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            static random_device rd;
            static mt19937 gen(rd());
            uniform_real_distribution<double> dis(0.0, 0.9);
            matrix[i][j] = round(dis(gen)*10)/10;
        }
    }

    vector<vector<double>> matrix_solution;
    for (int i = 1; i <= num_threads; i++) {
        auto start = chrono::steady_clock::now();
        matrix_solution = SolveSystemThreaded(matrix, i);
        auto end = chrono::steady_clock::now();
        auto diff = end - start;
        cout << i << ' ' << chrono::duration <double, milli> (diff).count() << " ms" <<
endl;
    }
    vector<double> solution = backSubstitution(matrix_solution);

    return 0;
}
```

Пример работы

```
artyom@artyom-Dell-G15-5510:~/Документы/vs code projects/OS_labs/lab2/build$ g++
../src/main.cpp -o main
```

```
artyom@artyom-Dell-G15-5510:~/Документы/vs code projects/OS_labs/lab2/build$ ./main
Enter size of matrix
5 6
Enter number of threads (from 1 to threads)
12
1 0.314654 ms
2 0.275278 ms
3 0.248599 ms
4 0.373349 ms
5 0.399895 ms
6 0.533775 ms
7 0.663467 ms
8 0.742482 ms
9 0.608536 ms
10 0.633899 ms
11 0.76433 ms
12 0.886892 ms
```

Вывод

При выполнении данной лабораторной работы я научился писать и отлаживать многопоточные программы. Познакомился с библиотекой thread.

Было интересно самому столкнуться с написанием многопоточных программ. Уверен, что это полезный навык, используемый в промышленной разработке.