**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МО ЭВМ**

# ОТЧЕТ

**по лабораторной работе №1**

**по дисциплине «Параллельные алгоритмы»**

**Тема: Основы работы с процессами и потоками**

Студент гр. 0303 _____ Бодунов П.А.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург

2023

**Цель работы.**

Изучение и практическое применение работы с процессами и потоками на языке C++.

**Задание.**

Выполнить умножение 2х матриц:

1)Входные матрицы вводятся из файла (или генерируются).

2)Результат записывается в файл.

**1.1.** Выполнить задачу, разбив её на 3 процесса. Выбрать механизм обмена данными между процессами.

1) Процесс 1: заполняет данными входные матрицы (читает из файла или генерирует их некоторым образом).

2) Процесс 2: выполняет умножение

3) Процесс 3: выводит результат

**1.2.1** Аналогично 1.1, используя потоки (std::threads)

**1.2.2** Разбить умножение на P потоков (“наивным” способом по строкам-столбцам).

Исследовать зависимость между количеством потоков, размерами входных данных и параметрами целевой вычислительной системы. Сформулировать ограничения.

**Выполнение работы.**

Создание матрицы происходит с помощью функции:

void generate_matrix(const std::string& filename, int rows, int columns) – на вход подаются имя файла в который записывается матрица, количество строк и столбцов. Элементы матрицы заполняются случайным целочисленными значениями в диапазоне [0, 99].

Чтение матрицы из файла осуществляется при помощи функции:

void read_matrix(vector<vector<int>>& matrix, string filename) – на вход подается ссылка на матрицу. Матрица будет заполняться значениями из файла filename.

Чтение двух матриц осуществляется при помощи функции:

void read_matrices(vector<vector<int>>& matrix1, vector<vector<int>>& matrix2) – на вход подается ссылки на матрицы. Матрицы будут заполняться значениями из файлов "/home/master/work/etu/PA/lab1/matrix_1.txt" и "/home/master/work/etu/PA/lab1/matrix_2.txt".

Умножение матриц для 1.1 и 1.2.1 происходит при помощи функции:

void multiply_matrices(vector<vector<int>>& matrix1, vector<vector<int>>& matrix2, vector<vector<int>>& matrix_res) – на вход подаются ссылки на 3 матрицы. matrix1 и matrix2, матрицы которые перемножаются, matrix_res – результат перемножения.

Умножение матриц для 1.2.2 происходит при помощи функции:

void multiply_matrices(vector<vector<int>>& matrix1, vector<vector<int>>& matrix2, vector<vector<int>>& matrix_res, int start, int end) – на вход подаются ссылки на 3 матрицы. matrix1 и matrix2, матрицы которые перемножаются, matrix_res – результат перемножения. А так же индексы start – начала и end – конца строк, которые будут перемножаться во время выполнения одного потока.

Запись результирующей матрицы осуществляется при помощи функции:

void write_result(vector<vector<int>> matrix_res, string filename) – передается результирующая матрица, а так же имя файла, в который запишется матрица.

**1.1 Реализация с помощью процессов.**

Умножение двух матриц при помощи процессов реализовано в файле process.cpp.

Разбиение на 3 процесса происходит при помощи функции fork() из библиотеки unistd.h, которая создаёт процессы-потомки. При помощи оператора switch-case происходит обработка каждого процесса: если значение PID равно 0 — выполняется код потомка, иначе — ожидается завершение работы другого процесса, при помощи функции wait(), в случае ошибки программа завершает работу.

**1.2.1 Реализация с помощью потоков.**

Умножение двух матриц при помощи потоков реализовано в файле threads.cpp.

Разбиение на потоки происходит при помощи создания конструктора класса thread из библиотеки thread, которая принимает необходимую функцию и её аргументы. Ожидание исполнения потока происходит при помощи метода join().

### 1.2.1 Реализация умножения матриц с помощью p потоков.

Умножение двух матриц при помощи p потоков реализовано в файле pthreads.cpp.

Строки матрицы разделяется на p частей так, чтобы каждому потоку соответствовало min_num_rows_thread или max_num_rows_thread, для равномерного распределения p потоков. Созданные потоки хранятся в векторе thread_vec, после ии выполнения вызывается метод join().

### 1.3 Исследовать зависимость между количеством потоков, размерами входных данных и параметрами целевой вычислительной системы.

Исследуем зависимость времени работы программы от количества потоков. Для этого возьмём постоянный размер матрицы равный 1000x1000. Результат зависимости времени умножения матриц от количества потоков представлен в табл. 1.

Таблица 1 — Зависимость времени работы программы от количества потоков

| Количество потоков P | Время работы программы, ms |
|---|---|
| 1 | 11053 |
| 7 | 4494 |
| 24 | 4173 |
| 100 | 4071 |
| 500 | 4031 |
| 1000 | 3992 |

Исходя из результатов таблицы, можно сделать вывод, что увеличение количества потоков ускоряет выполнение работы. Но с определенного момента увеличение количества потоков мало влияет на время выполнения программы.

Исследуем зависимость времени работы программы от размера входных данных, при количестве потоков равное 1. Результат зависимости времени умножения матриц от количества входных данных представлен в табл. 2.

Таблица 2 — Зависимость работы программы от входных данных

| Размер входной матрицы | Время работы программы, ms |
|---|---|
| 100x100 | 13 |
| 200x200 | 85 |
| 500x500 | 1349 |
| 1000x1000 | 10993 |

Исходя из результатов таблицы, можно сделать вывод, что увеличение размерности матрицы замедляет выполнение работы программы.

**Выводы.**

В процессе выполнения лабораторной работы были изучены и практически применены работы с процессами и потоками на языке C++. Были реализованы программы, соответствующие поставленным задачам, а именно:

1) Разбиение чтения, умножения и вывода матриц на 3 процесса

2) Разбиение чтения, умножения и вывода матриц на 3 потока

3) Разбиение чтения, умножения и вывода матриц на p потоков

При исследовании зависимостей времени работы от количества потоков, размера входных данных и параметров целевой вычислительной системы, было выявлено:

1) Увеличение количества потоков уменьшает время работы программы, но с определенного момента происходит "насыщение" и время работы программы не сильно уменьшается.

2) Увеличение количества элементов матрицы приводит к увеличению времени работы программы.

## ПРИЛОЖЕНИЕ А
## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: *processes.cpp*

```
#include <iostream>
#include <sys/wait.h>
#include <unistd.h>
#include <fstream>
#include <string>
#include <vector>

using namespace std;


void generate_matrix(const std::string& filename, int rows, int columns){
    ofstream file;
    file.open(filename);
    if (!file){
        cout << "Can't open file generate matrix\n";
        return;
    }
    for (int i = 0; i < rows; i++){
        for (int j = 0; j < columns; j++){
            file << rand() % 100 << ' ';
        }
        file << std::endl;
    }
    file.close();
}

void read_matrix(vector<vector<int>>& matrix, string filename){
    ifstream file(filename);
    if (!file){
        cout << "Can't open file read matrix";
        return;
    }

    string line;
    string delimiter = " ";
    vector<int> row;


    while (getline(file, line)){
        row.clear();
        size_t pos;

        while ((pos = line.find(delimiter)) != std::string::npos) {
            row.push_back(stoi(line.substr(0, pos)));
            line.erase(0, pos + delimiter.length());
        }
        matrix.push_back(row);
    }

    file.close();
}
```

```
void read_matrices(vector<vector<int>>& matrix1, vector<vector<int>>&
matrix2){
    read_matrix(matrix1, "/home/master/work/etu/PA/lab1/matrix_1.txt");
    read_matrix(matrix2, "/home/master/work/etu/PA/lab1/matrix_2.txt");
}

void multiply_matrices(vector<vector<int>>& matrix1, vector<vector<int>>&
matrix2, vector<vector<int>>& matrix_res){
    int row1 = matrix1.size();
    int col1 = matrix1[0].size();
    int row2 = matrix2.size();
    int col2 = matrix2[0].size();



    for(int i = 0; i < row1; i++){
        for(int j = 0; j < col2; j++){
            for(int k = 0; k < col1; k++)
                matrix_res[i][j] += matrix1[i][k] * matrix2[k][j];
        }
    }
}

void write_result(vector<vector<int>> matrix_res, string filename){
    ofstream file;
    file.open("/home/master/work/etu/PA/lab1/" + filename);
    for (int i = 0; i < matrix_res.size(); i++){
        for(int j = 0; j < matrix_res[i].size(); j++){
            file << matrix_res[i][j] << ' ';
        }
        file << std::endl;
    }
    file.close();
}

int main(){
    generate_matrix("/home/master/work/etu/PA/lab1/matrix_1.txt", 3, 3);
    generate_matrix("/home/master/work/etu/PA/lab1/matrix_2.txt", 3, 3);

    vector<vector<int>> matrix1;
    vector<vector<int>> matrix2;

    read_matrices(matrix1, matrix2);

    vector<vector<int>> matrix_res(matrix1.size(),
vector<int>(matrix2[0].size(), 0));

    int sum_pid;

    switch(sum_pid = fork()){
        case -1:
            exit(-1);

        case 0:
            multiply_matrices(matrix1, matrix2, matrix_res);

            int write_pid;
            switch(write_pid = fork()){
```

```
                case -1:
                    exit(-1);
                case 0:
                    write_result(matrix_res, "processes_res.txt");
                    exit(1);
                default:
                    wait(&write_pid);
            }
            exit(1);
        default:
            wait(&sum_pid);
    }
}
```

Название файла: *threads.h*

```
#include <iostream>
#include <sys/wait.h>
#include <unistd.h>
#include <fstream>
#include <string>
#include <vector>

#include <thread>
#include <chrono>

using namespace std;


void generate_matrix(const std::string& filename, int rows, int columns){
    ofstream file;
    file.open(filename);
    if (!file){
        cout << "Can't open file generate matrix\n";
        return;
    }
    for (int i = 0; i < rows; i++){
        for (int j = 0; j < columns; j++){
            file << rand() % 100 << ' ';
        }
        file << std::endl;
    }
    file.close();
}

void read_matrix(vector<vector<int>>& matrix, string filename){
    ifstream file(filename);
    if (!file){
        cout << "Can't open file read matrix";
        return;
    }

    string line;
    string delimiter = " ";
    vector<int> row;
```

```
    while (getline(file, line)){
        row.clear();
        size_t pos;

        while ((pos = line.find(delimiter)) != std::string::npos) {
            row.push_back(stoi(line.substr(0, pos)));
            line.erase(0, pos + delimiter.length());
        }
//        row.push_back(stoi(line));
        matrix.push_back(row);
    }

    file.close();
}

void read_matrices(vector<vector<int>>& matrix1, vector<vector<int>>&
matrix2){
    read_matrix(matrix1, "/home/master/work/etu/PA/lab1/matrix_1.txt");
    read_matrix(matrix2, "/home/master/work/etu/PA/lab1/matrix_2.txt");
}


void multiply_matrices(vector<vector<int>>& matrix1, vector<vector<int>>&
matrix2, vector<vector<int>>& matrix_res){
    int row1 = matrix1.size();
    int col1 = matrix1[0].size();
    int row2 = matrix2.size();
    int col2 = matrix2[0].size();



    for(int i = 0; i < row1; i++){
        for(int j = 0; j < col2; j++){
            for(int k = 0; k < col1; k++)
                matrix_res[i][j] += matrix1[i][k] * matrix2[k][j];
        }
    }
}

void write_result(vector<vector<int>> matrix_res, string filename){
    ofstream file;
    file.open("/home/master/work/etu/PA/lab1/" + filename);
    for (int i = 0; i < matrix_res.size(); i++){
        for(int j = 0; j < matrix_res[i].size(); j++){
            file << matrix_res[i][j] << ' ';
        }
        file << std::endl;
    }
    file.close();
}

int main(){
    //    generate_matrix("/home/master/work/etu/PA/lab1/matrix_1.txt",
1000, 1000);
    //    generate_matrix("/home/master/work/etu/PA/lab1/matrix_2.txt",
1000, 1000);
```

```
    vector<vector<int>> matrix1;
    vector<vector<int>> matrix2;

    thread thread_read(read_matrices, std::ref(matrix1),
std::ref(matrix2));
    thread_read.join();

    vector<vector<int>> matrix_res(matrix1.size(),
vector<int>(matrix2[0].size(), 0));

    thread thread_mult(multiply_matrices, ref(matrix1), ref(matrix2),
ref(matrix_res));
    thread_mult.join();

    thread threat_write(write_result, ref(matrix_res),
"threads_res.txt");
    threat_write.join();
}
```

Название файла: *pthreads.h*

```
#include <iostream>
#include <sys/wait.h>
#include <unistd.h>
#include <fstream>
#include <string>
#include <vector>

#include <thread>
#include <chrono>

using namespace std;


void generate_matrix(const std::string& filename, int rows, int columns){
    ofstream file;
    file.open(filename);
    if (!file){
        cout << "Can't open file generate matrix\n";
        return;
    }
    for (int i = 0; i < rows; i++){
        for (int j = 0; j < columns; j++){
            file << rand() % 100 << ' ';
        }
        file << std::endl;
    }
    file.close();
}

void read_matrix(vector<vector<int>>& matrix, string filename){
    ifstream file(filename);
    if (!file){
        cout << "Can't open file read matrix";
        return;
    }

    string line;
```

```
    string delimiter = " ";
    vector<int> row;


    while (getline(file, line)){
        row.clear();
        size_t pos;

        while ((pos = line.find(delimiter)) != std::string::npos) {
            row.push_back(stoi(line.substr(0, pos)));
            line.erase(0, pos + delimiter.length());
        }
        matrix.push_back(row);
    }

    file.close();
}

void read_matrices(vector<vector<int>>& matrix1, vector<vector<int>>&
matrix2){
    read_matrix(matrix1, "/home/master/work/etu/PA/lab1/matrix_1.txt");
    read_matrix(matrix2, "/home/master/work/etu/PA/lab1/matrix_2.txt");
}

void multiply_matrices(vector<vector<int>>& matrix1, vector<vector<int>>&
matrix2, vector<vector<int>>& matrix_res, int start, int end){
    int row1 = matrix1.size();
    int col1 = matrix1[0].size();
    int row2 = matrix2.size();
    int col2 = matrix2[0].size();



    for(int i = start; i <= end; i++){
        for(int j = 0; j < col2; j++){
            for(int k = 0; k < col1; k++)
                matrix_res[i][j] += matrix1[i][k] * matrix2[k][j];
        }
    }
}

void write_result(vector<vector<int>> matrix_res, string filename){
    ofstream file;
    file.open("/home/master/work/etu/PA/lab1/" + filename);
    for (int i = 0; i < matrix_res.size(); i++){
        for(int j = 0; j < matrix_res[i].size(); j++){
            file << matrix_res[i][j] << ' ';
        }
        file << std::endl;
    }
    file.close();
}

int main(){
//    generate_matrix("/home/master/work/etu/PA/lab1/matrix_1.txt", 1000,
1000);
//    generate_matrix("/home/master/work/etu/PA/lab1/matrix_2.txt", 1000,
1000);
```

```
    vector<vector<int>> matrix1;
    vector<vector<int>> matrix2;
    int num_threads = 1;

    auto start_time = chrono::high_resolution_clock::now();

    thread thread_read(read_matrices, std::ref(matrix1),
std::ref(matrix2));
    thread_read.join();

    vector<vector<int>> matrix_res(matrix1.size(),
vector<int>(matrix2[0].size(), 0));


    vector<thread> thread_vec;
    int min_num_rows_thread = matrix1.size() / num_threads;
    int max_num_rows_thread = min_num_rows_thread + 1;
    int num_threads_with_max_rows = matrix1.size() % num_threads;

    for (int i = 0; i < num_threads; i++){
        if (i < num_threads_with_max_rows)
            thread_vec.push_back(thread(multiply_matrices, ref(matrix1),
ref(matrix2), ref(matrix_res),
            i * max_num_rows_thread, (i + 1) * max_num_rows_thread));
        else
            thread_vec.push_back(thread(multiply_matrices, ref(matrix1),
ref(matrix2), ref(matrix_res),
            num_threads_with_max_rows * max_num_rows_thread + (i -
num_threads_with_max_rows) * min_num_rows_thread,
            num_threads_with_max_rows * max_num_rows_thread + (i -
num_threads_with_max_rows + 1) * min_num_rows_thread - 1));

    }

    for (int i = 0; i < thread_vec.size(); i++)
        thread_vec[i].join();

    thread threat_write(write_result, ref(matrix_res),
"pthreads_res.txt");
    threat_write.join();

    auto stop_time = chrono::high_resolution_clock::now();
    auto duration = chrono::duration_cast<chrono::milliseconds>(stop_time
- start_time);

    cout << "Duration " << num_threads << " threads: "<< duration.count()
<< " ms" << endl;
}
```