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 _____ Середенков А.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 generate_matrix(), реализованной в файле matrix_operations.h. На вход функция получает матрицу, представляющую из себя вектор векторов и её размерность. В процессе выполнения программы элементы матрицы заполняются случайным целочисленными значениями в диапазоне [-9, 9].

Реализация умножения матриц при помощи процессов.

Умножение двух матриц при помощи процессов реализовано в файле process.cpp. В самом начале программа спрашивает у пользователя количество строк и столбцов матрицы.

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

Реализация умножения матриц при помощи потоков.

Умножение двух матриц при помощи потоков реализовано в файле threads.cpp. В самом начале программа спрашивает у пользователя количество строк и столбцов матрицы.

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().

Реализация умножения матриц при помощи p-потоков.

Умножение двух матриц при помощи потоков реализовано в файле pthreads.cpp. В самом начале программа спрашивает у пользователя количество строк, столбцов матрицы и количество потоков.

Для равномерного разбиения на p-потоков, матрица разделяется на p частей по строкам первой матрицы. Все созданные потоки хранятся в векторе threads, после их инициализации итерационно вызывается метод join().

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	47215
10	6284
100	6481
1000	6459
2000	31598

Исходя из результатов таблицы, можно сделать вывод, что увеличение количества потоков ускоряет выполнение работы, однако при значениях превышающих размерность матрицы время начинает многократно возрастать. Следовательно максимальное значение потоков не должно превышать размерность матрицы.

Исследуем зависимость времени работы программы от размера входных данных. 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
2x2	1662
5x5	1764
10x10	1608
100x100	2060
1000x1000	40180

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

заметить, что увеличение размерности в пределах одного порядка не сильно увеличивает время работы.

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 создания, умножения и вывода матриц на 3 процесса

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

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

При исследовании зависимостей времени работы от количества потоков, размера входных данных и параметров целевой вычислительной системы были получены следующие результаты:

1) Увеличение количества потоков не всегда уменьшает время работы программы, необходимо оптимальное число потоков, чтобы не тратить время на их инициализацию.

2) Увеличение количества входных данных приводит к увеличению времени работы программы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: matrix_operations.h

```
#include <iostream>
#include <ctime>
#include <fstream>

using namespace std;

void generate_matrix(vector<vector<int>>& matrix, int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            //matrix[i][j] = rand() % 10000 - 5000;
            matrix[i][j] = rand() % 19 - 9;
            cout<< matrix[i][j] << " ";
        }
        cout<<endl;
    }
    cout<< "======"<< endl;
}


vector<vector<int>> multiply_matrices(vector<vector<int>>& a, vector<vector<int>>& b, int m, int n) {
    vector<vector<int>> c(m, vector<int>(m));
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            c[i][j] = 0;
            for (int k = 0; k < n; k++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    return c;
}
```

```
void print_res(vector<vector<int>>& result_matrix, int m){
    ofstream result;
    result.open("matrix_result.txt");
    //cout << "The product of the two matrices is:\n";
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            //cout << result_matrix[i][j] << " ";
            result << result_matrix[i][j] << " ";
        }
        //cout << endl;
        result << endl;
    }
}
```

Название файла: process.cpp

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

#include "matrix_operations.h"

using namespace std;

int main(){
    int n = 0, m = 0;
    int id_mult, id_write;
    cout<<"Введите размер матриц: ";
    cin>>m>>n;
    vector<vector<int>> first_matrix(m, vector<int>(n));
    vector<vector<int>> second_matrix(n, vector<int>(m));
    vector<vector<int>> result_matrix(m, vector<int>(m));

    generate_matrix(first_matrix, m, n);
    generate_matrix(second_matrix, n, m);
```

```
        id_mult = fork();
        switch(id_mult){
            case -1:
                exit(-1);
            case 0:
                    result_matrix = multiply_matrices(first_matrix,
second_matrix, m, n);
                id_write = fork();
                switch (id_write){
                    case -1:
                        exit(-1);
                    case 0:
                        print_res(result_matrix, m);
                        exit(0);
                    default:
                        wait(&id_write);
                }
                exit(0);
            default:
                wait(&id_mult);
        }
        return 0;
    }
```

Название файла: threads.h

```
#include <iostream>
#include <vector>
#include <chrono>
#include <thread>

#include "matrix_operations.h"

using namespace std;


void create_mat(vector<vector<int>>& first_matrix,
vector<vector<int>>& second_matrix, int m, int n){
```

```
        generate_matrix(first_matrix, m, n);
        generate_matrix(second_matrix, n, m);
    }

    void count_mat(vector<vector<int>>& c,vector<vector<int>>& a,
vector<vector<int>>& b, int m, int n){
        c = multiply_matrices(a, b, m, n);
    }

    int main(){
        int n = 0, m = 0;
        auto start = chrono::high_resolution_clock::now();

        cout<<"Введите размер матриц: ";
        cin>>m>>n;
        vector<vector<int>> first_matrix(m, vector<int>(n));
        vector<vector<int>> second_matrix(n, vector<int>(m));
        vector<vector<int>> result_matrix(m, vector<int>(m));

                thread make_matrix(create_mat, ref(first_matrix),
ref(second_matrix), ref(m), ref(n));
        make_matrix.join();

                thread count_matrix(count_mat, ref(result_matrix),
ref(first_matrix), ref(second_matrix), ref(m), ref(n));
        count_matrix.join();

        thread print_matrix(print_res, ref(result_matrix), ref(m));
        print_matrix.join();
        auto stop = chrono::high_resolution_clock::now();
                                    auto duration =
chrono::duration_cast<chrono::milliseconds>(stop - start);
              cout << "Продолжительность работы программы: "<<
duration.count()<< " ms" << endl;

        return 0;
    }
```

Название файла: pthreads.h

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

#include "matrix_operations.h"

int n = 0, m = 0;

void multiply(int start, int end, const std::vector<std::vector<int>>& A, const std::vector<std::vector<int>>& B, std::vector<std::vector<int>>& C) {
    for (int i = start; i < end; i++) {
        for (int j = 0; j < m; j++) {
            C[i][j] = 0;
            for (int k = 0; k < n; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

void create_mat(vector<vector<int>>& first_matrix, vector<vector<int>>& second_matrix, int m, int n){
    generate_matrix(first_matrix, m, n);
    generate_matrix(second_matrix, n, m);
}

int main() {
    cout<<"Введите размер матриц: ";
    cin>>m>>n;

    int THREADS = 0;
    cout<<"Введите количество потоков: ";
    cin>>THREADS;
    vector<vector<int>> first_matrix(m, vector<int>(n));
```

```
        vector<vector<int>> second_matrix(n, vector<int>(m));
        vector<vector<int>> result_matrix(m, vector<int>(m));

        auto start = chrono::high_resolution_clock::now();
                thread   make_matrix(create_mat,   ref(first_matrix),
ref(second_matrix), ref(m), ref(n));
        make_matrix.join();

        std::vector<std::thread> threads;
        int chunk_size = m / THREADS;
        for (int i = 0; i < THREADS; i++) {
            int start = i * chunk_size;
            int end = (i == THREADS - 1) ? m : (i + 1) * chunk_size;
                        threads.emplace_back(multiply,   start,   end,
std::ref(first_matrix),                         std::ref(second_matrix),
std::ref(result_matrix));
        }

        for (auto& thread : threads) {
            thread.join();
        }

        print_res(result_matrix, m);
        auto stop = chrono::high_resolution_clock::now();
                                        auto         duration         =
chrono::duration_cast<chrono::milliseconds>(stop - start);
                cout   <<   "Продолжительность   работы   программы:   "<<
duration.count()<< " ms" << endl;

        return 0;
    }
```