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 Морозов А.Ю.

Преподаватель Сергеева Е.И.

Санкт-Петербург
2023

Цель работы.

Изучение основ работы с процессами и потоками.

Задание.

Лабораторная состоит из трех подзадач, которые выполняют *одинаковую задачу* с использованием процессов или потоков. Требуется выполнить умножение двух матриц. Входные матрицы вводятся из файла (или генерируются). Результат записывается в файл.

Подзадача 1:

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

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

Процесс 2: выполняет умножение

Процесс 3: выводит результат

Подзадача 2:

Аналогично 1.1, используя потоки (std::threads).

Подзадача 3:

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

В отчёте:

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

Выполнение работы.

Структура проекта:

- data – папка, в которую записываются текстовые файлы, содержащие две матрицы-множителя и результат их перемножения.

- matrix_generator – папка, содержащая описание и реализацию функций для генерации матриц-множителей размерностью m * n, заполненных случайными числами от -10 до 10 и их записи в соответствующие файлы.
- matrix_reader – папка, содержащая описание и реализацию функций для считывания матриц-множителей из файлов и их представления в виде двумерного вектора целых чисел.
- result_writer – папка, содержащая описание и реализацию функций для записи результата умножения в соответствующий файл.
- main_matrix_generator.cpp – главный файл, в котором происходит генерация матриц.
- main_processes.cpp – главный файл, в котором задача решается с помощью процессов.
- main_threads.cpp – главный файл, в котором задача решается с помощью потоков.
- main_parallel.cpp – главный файл, в котором задача перемножения матриц распараллеливается между несколькими потоками.
- Makefile – удобное средство сборки и очистки проекта.

Описание подзадач:

1. Три процесса.

В качестве механизма обмена данными была выбрана разделяемая память. Первый процесс считывает матрицы с файлов, превращает в векторы и записывает в разделяемую область памяти. Второй процесс считывает матрицы из разделяемой области памяти, производит перемножение и записывает результирующую матрицу в разделяемую область памяти. Третий процесс читает результат и пишет в соответствующий файл.

2. Три потока.

Аналогично реализации подзадачи 1 (см. выше), за исключением того, что данные больше не пишутся в разделяемую область памяти, а изменяются по ссылкам.

3. Параллельное умножение.

Аналогично реализации подзадачи 2 (см. выше), за исключением того, что над умножением матриц трудятся несколько потоков, между которыми равномерно распределены элементы результирующей матрицы.

Исследование.

Таблица 1 – Связь кол-ва потоков и времени работы программы.

Кол-во потоков	Время работы программы
1	7.020 с.
2	4.008 с.
4	3.401 с.

Размерность входных матриц: 1000 * 1000.

Таблица 2 – Связь размера входных данных и времени работы программы.

Размер входных данных	Время работы программы
10 * 10	2 мс.
250 * 250	64 мс.
800 * 800	1.965 с.
1500 * 1500	12.056 с.

Количество потоков на умножение: 2.

Выводы.

В ходе выполнения лабораторной работы были получены знания основ работы с процессами и потоками. Результатом выполнения лабораторной работы стали реализованные программы, позволяющие посчитать произведение матриц при помощи процессов, потоков и потоков, параллельно выполняющих вычисления.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: matrix_generator.hpp

```
#include <vector>
#include <fstream>

void generate_multipliers(
   int first_height,
   int first_width,
   int second_height,
   int second_width
);
void generate_first_multiplier(int height, int width);
void generate_second_multiplier(int height, int width);
std::vector<std::vector<int>> generate_matrix_body(int height, int
width);
```

Название файла: matrix_generator.cpp

```
#include "matrix_generator.hpp"

void generate_first_multiplier(int height, int width) {
   std::vector<std::vector<int>> matrix =
generate_matrix_body(height, width);
   std::ofstream out;
   out.open("./data/first_multiplier.txt");
   if (out.is_open())
   {
       out << height << " " << width << std::endl;
       for (int i = 0; i < height; i++) {
           for (int j = 0; j < width - 1; j++) {
               out << matrix[i][j] << " ";
           }
           out << matrix[i][width - 1] << std::endl;
       }
   }
   out.close();
}

void generate_second_multiplier(int height, int width) {
   std::vector<std::vector<int>> matrix =
generate_matrix_body(height, width);
   std::ofstream out;
   out.open("./data/second_multiplier.txt");
   if (out.is_open())
   {
       out << height << " " << width << std::endl;
       for (int i = 0; i < height; i++) {
           for (int j = 0; j < width - 1; j++) {
               out << matrix[i][j] << " ";
```

```
            }
            out << matrix[i][width - 1] << std::endl;
        }
    }
    out.close();
}

std::vector<std::vector<int>> generate_matrix_body(int height, int
width) {
    std::vector<std::vector<int>> matrix(height,
std::vector<int>(width));
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            matrix[i][j] = -10 + rand() % 21;
        }
    }
    return matrix;
}

void generate_multipliers(int first_height, int first_width,
                          int second_height, int second_width
) {
    srand(time(NULL));
    generate_first_multiplier(first_height, first_width);
    generate_second_multiplier(second_height, second_width);
}
```

Название файла: matrix_reader.hpp

```
#include <vector>
#include <fstream>

std::vector<std::vector<int>> read_first_multiplier();
std::vector<std::vector<int>> read_second_multiplier();
```

Название файла: matrix_reader.cpp

```
#include "matrix_reader.hpp"

std::vector<std::vector<int>> read_first_multiplier() {
    std::vector<std::vector<int>> matrix;
    std::ifstream in("./data/first_multiplier.txt");
    if (in.is_open())
    {
        int height, width;
        in >> height;
        in >> width;
        std::vector<std::vector<int>> tmp(height,
std::vector<int>(width));
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                in >> tmp[i][j];
```

```
            }
        }
        matrix = tmp;
    }
    in.close();
    return matrix;
}

std::vector<std::vector<int>> read_second_multiplier() {
    std::vector<std::vector<int>> matrix;
    std::ifstream in("./data/second_multiplier.txt");
    if (in.is_open())
    {
        int height, width;
        in >> height;
        in >> width;
        std::vector<std::vector<int>> tmp(height,
std::vector<int>(width));
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                in >> tmp[i][j];
            }
        }
        matrix = tmp;
    }
    in.close();
    return matrix;
}
```

Название файла: result_writer.hpp

```
#include <vector>
#include <fstream>

void write_result(std::vector<std::vector<int>> result_matrix);
```

Название файла: result_writer.cpp

```
#include "result_writer.hpp"
void write_result(std::vector<std::vector<int>> result_matrix) {
     std::ofstream out;
     out.open("./data/result.txt");
     if (out.is_open())
     {
     int height = result_matrix.size();
     int width = result_matrix[0].size();
     for (int i = 0; i < height; i++) {
```

```
            for (int j = 0; j < width - 1; j++) {
                 out << result_matrix[i][j] << " ";
            }
            out << result_matrix[i][width - 1] << std::endl;
      }
      }
      out.close();
}
```

Название файла: main_matrix_generator.cpp

```
#include "matrix_generator/matrix_generator.hpp"

int main() {

   int first_height = 10;
   int first_width = 10;
   int second_height = 10;
   int second_width = 10;

   generate_multipliers(
       first_height,
       first_width,
       second_height,
       second_width
   );

   return 0;
}
```

Название файла: main_processes.cpp

```
#include "matrix_reader/matrix_reader.hpp"
#include "result_writer/result_writer.hpp"
#include <unistd.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <cmath>
#include <iostream>

void* get_shared_memory(){
   int id = shmget(1, 4194304, 0644 | IPC_CREAT);
   void* pointer = shmat(id, nullptr, 0);
   return pointer;
}

int* write_to_shared_memory(int* shared_memory,
std::vector<std::vector<int>> matrix) {
```

```
    int height = matrix.size();
    int width = matrix[0].size();
    *shared_memory = height;
    shared_memory++;
    *shared_memory = width;
    shared_memory++;
    for (int i = 0; i < height; ++i) {
        for (int j = 0; j < width; ++j) {
            *shared_memory = matrix[i][j];
            shared_memory++;
        }
    }
    return shared_memory;
}

std::vector<std::vector<int>> read_from_shared_memory(int*
shared_memory) {
    int height = *shared_memory;
    shared_memory++;
    int width = *shared_memory;
    shared_memory++;
    std::vector<std::vector<int>> matrix(height,
std::vector<int>(width));
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            matrix[i][j] = *shared_memory;
            shared_memory++;
        }
    }
    return matrix;
}

void read_process() {
    std::vector<std::vector<int>> first_multiplier =
read_first_multiplier();
    std::vector<std::vector<int>> second_multiplier =
read_second_multiplier();
    int* shared_memory = (int*)get_shared_memory();
    shared_memory = write_to_shared_memory(shared_memory,
first_multiplier);
    write_to_shared_memory(shared_memory, second_multiplier);
}

void multiplication_process() {
    int* shared_memory = (int*)get_shared_memory();
    std::vector<std::vector<int>> first_multiplier =
read_from_shared_memory(shared_memory);
    shared_memory += (2 + first_multiplier.size() *
first_multiplier[0].size());
    std::vector<std::vector<int>> second_multiplier =
read_from_shared_memory(shared_memory);
    shared_memory -= (2 + first_multiplier.size() *
first_multiplier[0].size());
```

```
    int height = first_multiplier.size();
    int width = second_multiplier[0].size();
    int equal_line = second_multiplier.size();
    std::vector<std::vector<int>> result_matrix(height,
std::vector<int>(width));
    int tmp_sum;
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            tmp_sum = 0;
            for (int n = 0; n < equal_line; n++) {
                tmp_sum += (first_multiplier[i][n] *
second_multiplier[n][j]);
            }
            result_matrix[i][j] = tmp_sum;
        }
    }
    write_to_shared_memory(shared_memory, result_matrix);
}

void write_process() {
    int* shared_memory = (int*)get_shared_memory();
    std::vector<std::vector<int>> result_matrix =
read_from_shared_memory(shared_memory);
    write_result(result_matrix);
}


int main() {

    pid_t read_pid = fork();
    switch(read_pid){
        case 0:
            read_process();
            exit(0);
        default:
            wait(&read_pid);
    }

    pid_t multiplication_pid = fork();
    switch(multiplication_pid){
        case 0:
            multiplication_process();
            exit(0);
        default:
            wait(&multiplication_pid);
    }

    pid_t write_pid = fork();
    switch(write_pid){
        case 0:
            write_process();
            exit(0);
        default:
```

```
            wait(&write_pid);
    }

    return 0;
}
```

Название файла: main_threads.cpp

```
#include <thread>
#include "matrix_reader/matrix_reader.hpp"
#include "result_writer/result_writer.hpp"

void read_task(
    std::vector<std::vector<int>> &first_multiplier,
    std::vector<std::vector<int>> &second_multiplier
) {
    first_multiplier = read_first_multiplier();
    second_multiplier = read_second_multiplier();
}

void multiplication_task(
    std::vector<std::vector<int>> first_multiplier,
    std::vector<std::vector<int>> second_multiplier,
    std::vector<std::vector<int>> &result_matrix
) {
    int height = first_multiplier.size();
    int width = second_multiplier[0].size();
    int equal_line = second_multiplier.size();
    std::vector<std::vector<int>> tmp_matrix(height,
std::vector<int>(width));
    int tmp_sum;
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            tmp_sum = 0;
            for (int n = 0; n < equal_line; n++) {
                tmp_sum += (first_multiplier[i][n] *
second_multiplier[n][j]);
            }
            tmp_matrix[i][j] = tmp_sum;
        }
    }
    result_matrix = tmp_matrix;
}

void write_task(std::vector<std::vector<int>> result_matrix) {
    write_result(result_matrix);
}


int main() {
    std::vector<std::vector<int>> first_multiplier;
    std::vector<std::vector<int>> second_multiplier;
```

```
    std::vector<std::vector<int>> result_matrix;

    std::thread reader(
        read_task,
        std::ref(first_multiplier),
        std::ref(second_multiplier)
    );
    reader.join();

    std::thread multiplicator(
        multiplication_task,
        first_multiplier,
        second_multiplier,
        std::ref(result_matrix)
    );
    multiplicator.join();

    std::thread writer(write_task, result_matrix);
    writer.join();

    return 0;
}
```

Название файла: main_parallel.cpp

```
#include <thread>
#include <cmath>
#include <chrono>
#include <iostream>
#include "matrix_reader/matrix_reader.hpp"
#include "result_writer/result_writer.hpp"

void read_task(
    std::vector<std::vector<int>> &first_multiplier,
    std::vector<std::vector<int>> &second_multiplier
) {
    first_multiplier = read_first_multiplier();
    second_multiplier = read_second_multiplier();
}

void multiplication_task(
    std::vector<std::vector<int>> first_multiplier,
    std::vector<std::vector<int>> second_multiplier,
    int start,
    int elem_amount,
    std::vector<std::vector<int>> &result_matrix
) {
    int height = first_multiplier.size();
    int width = second_multiplier[0].size();
    int equal_line = second_multiplier.size();
    int tmp_sum, i, j;
    for (int l = start; l < start + elem_amount; l++) {
        tmp_sum = 0;
        i = l / width;
        j = l - i * width;
```

```
        for (int n = 0; n < equal_line; n++) {
            tmp_sum += (first_multiplier[i][n] *
second_multiplier[n][j]);
        }
        result_matrix[i][j] = tmp_sum;
    }
}

void write_task(std::vector<std::vector<int>> result_matrix) {
    write_result(result_matrix);
}


int main() {

    auto start_programm = std::chrono::high_resolution_clock::now();

    std::vector<std::vector<int>> first_multiplier;
    std::vector<std::vector<int>> second_multiplier;

    std::thread reader(
        read_task,
        std::ref(first_multiplier),
        std::ref(second_multiplier)
    );
    reader.join();

    int thread_amount = 4;
    int height = first_multiplier.size();
    int width = second_multiplier[0].size();
    std::vector<std::vector<int>> result_matrix(height,
std::vector<int>(width));
    int elem_amount = height * width;
    int elem_per_thread = ceil((double)elem_amount / thread_amount);
    int last_thread_elem = elem_amount - (thread_amount - 1) *
elem_per_thread;
    std::vector<std::thread> threads;
    int tmp_start = 0;
    for (int i = 0; i < thread_amount - 1; i++) {
        threads.push_back(std::thread(
            multiplication_task,
            first_multiplier,
            second_multiplier,
            tmp_start,
            elem_per_thread,
            std::ref(result_matrix)
        ));
        tmp_start += elem_per_thread;
    }
    threads.push_back(std::thread(
        multiplication_task,
        first_multiplier,
        second_multiplier,
        tmp_start,
        last_thread_elem,
        std::ref(result_matrix)
    ));
```

```
    for (int i = 0; i < threads.size(); i++) {
        threads[i].join();
    }

    std::thread writer(write_task, result_matrix);
    writer.join();

    auto end_programm = std::chrono::high_resolution_clock::now();
    auto duration =
std::chrono::duration_cast<std::chrono::milliseconds>(end_programm -
start_programm);
    std::cout << duration.count() << std::endl;

    return 0;
}
```