Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**УПРАВЛЕНИЕ ПОТОКАМИ В ОС**

Студент: Мамонтов Егор Олегович

Группа: М8О–212Б–22

Вариант: 20

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Управление потоками в ОС
* Обеспечение синхронизации между потоками

## Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы Unix. Ограничение максимального количества потоков, работающих в один момент времени, должно быть задано ключом запуска вашей программы.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

**Вариант 20:** Дан массив координат (x, y, z). Необходимо найти три точки, которые образуют треугольник максимальной площади

**Общие сведения о программе**

Программа компилируется из файла main.cpp. Также используется заголовочные файлы: <iostream>, <cmath>, <pthread.h>, <vector>, <cstdint>, <chrono>. Программы собраны с помощью Cmake. В программе используются следующие системные вызовы:

1. **pthread\_create** – создание потока.
2. **pthread\_join** – ожидание потока и его завершение.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы pthread\_create, pthread\_join, pthread\_detach, pthread\_mutex.
2. Написать структуры TriangleData и Point для удобства хранения данных о треугольнике и его точках.
3. Написать файл main.cpp, в котором будет осуществляться перебор всех возможных точек треугольника с нахождением их площади в многопоточном режиме.

**Основные файлы программы**

**main.cpp:**

#include <iostream>

#include <cmath>

#include <pthread.h>

#include <vector>

#include <cstdint>

#include <chrono>

struct Point {

int x, y, z;

};

struct TriangleData {

float max\_S;

unsigned int pt\_count;

unsigned int thread\_count;

int\*\* array;

};

/\* Division with rounding up \*/

int div\_up(int x, int y) {

return (x + y - 1) / y;

}

/\* Get triangle area \*/

void square(float& S, Point p[3]) {

int opr\_matr1 = (1 \* p[1].x \* p[2].y) +

(p[1].y \* p[0].x \* 1) +

(p[2].x \* p[0].y \* 1) -

(p[0].y \* p[1].x \* 1) -

(p[1].y \* p[2].x \* 1) -

(p[2].y \* p[0].x \* 1);

int opr\_matr2 = (1 \* p[1].x \* p[2].z) +

(p[1].z \* p[0].x \* 1) +

(p[2].x \* p[0].z \* 1) -

(p[0].z \* p[1].x \* 1) -

(p[1].z \* p[2].x \* 1) -

(p[2].z \* p[0].x \* 1);

int opr\_matr3 = (1 \* p[1].y \* p[2].z) +

(p[1].z \* p[0].y \* 1) +

(p[2].y \* p[0].z \* 1) -

(p[0].z \* p[1].y \* 1) -

(p[1].z \* p[2].y \* 1) -

(p[2].z \* p[0].y \* 1);

S = 0.5 \* pow(abs(pow(opr\_matr1, 2) + pow(opr\_matr2, 2) + pow(opr\_matr3, 2)), 0.5);

}

/\* Function for working with threads \*/

void\* perebor(void\* a) {

TriangleData\* t\_data = (TriangleData\*)a;

float S = 0;

float max\_S = 0;

Point p[3];

int pt\_c = t\_data->pt\_count;

int thr\_c = t\_data->thread\_count;

int thr\_num = t\_data->max\_S;

int x = (div\_up(pt\_c, thr\_c)) \* (thr\_num + 1);

if (x >= pt\_c) {

x = pt\_c - 1;

}

for (unsigned int i = 0; i < pt\_c; ++i) {

p[0].x = t\_data->array[i][0];

p[0].y = t\_data->array[i][1];

p[0].z = t\_data->array[i][2];

for (unsigned int j = 0; j < pt\_c; ++j) {

p[1].x = t\_data->array[j][0];

p[1].y = t\_data->array[j][1];

p[1].z = t\_data->array[j][2];

for (unsigned int k = (div\_up(pt\_c, thr\_c)) \* thr\_num; k < x; ++k) {

p[2].x = t\_data->array[k][0];

p[2].y = t\_data->array[k][1];

p[2].z = t\_data->array[k][2];

square(S, p);

if (S > max\_S) {

max\_S = S;

}

}

}

}

return reinterpret\_cast<void\*>(new float(max\_S));

}

int main() {

unsigned int pt\_count, thread\_count;

float\* max\_S = nullptr;

float maximum = 0;

std::cout << "Count threads: ";

std::cin >> thread\_count;

std::cout << "Count points: ";

std::cin >> pt\_count;

int\*\* array = new int\*[pt\_count];

for (unsigned int i = 0; i < pt\_count; ++i) {

array[i] = new int[3];

}

for (unsigned int i = 0; i < pt\_count; i++) {

for (unsigned int j = 0; j < 3; j++) {

std::cin >> array[i][j];

}

}

std::cout << "Start!" << std::endl;

auto start\_time = std::chrono::high\_resolution\_clock::now(); // Start the timer

/\* Work with threads \*/

pthread\_t thread\_id[thread\_count];

for (int n = 0; n < thread\_count; n++) {

TriangleData t\_data{0, pt\_count, thread\_count, array};

t\_data.max\_S = n;

pthread\_create(&thread\_id[n + 1], nullptr, perebor, &t\_data);

}

for (int n = 0; n < thread\_count; n++) { // Correct loop bounds

pthread\_join(thread\_id[n + 1], reinterpret\_cast<void\*\*>(&max\_S));

if (\*max\_S >= maximum) {

maximum = \*max\_S;

}

delete max\_S; // Release memory

}

auto end\_time = std::chrono::high\_resolution\_clock::now(); // Stop the timer

auto duration = std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time);

std::cout << "Итоговое время: " << duration.count() << " миллисекунд" << std::endl;

std::cout << "Итоговый максимум: " << maximum << std::endl;

/\* Delete array \*/

for (unsigned int i = 0; i < pt\_count; ++i) {

delete[] array[i];

}

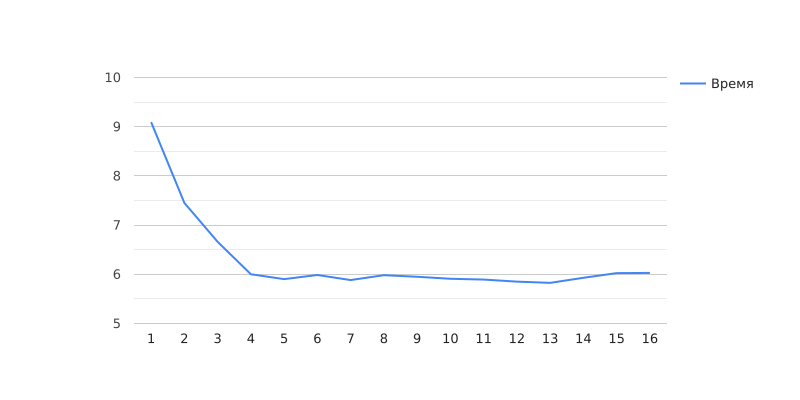
delete[] array;

return 0;

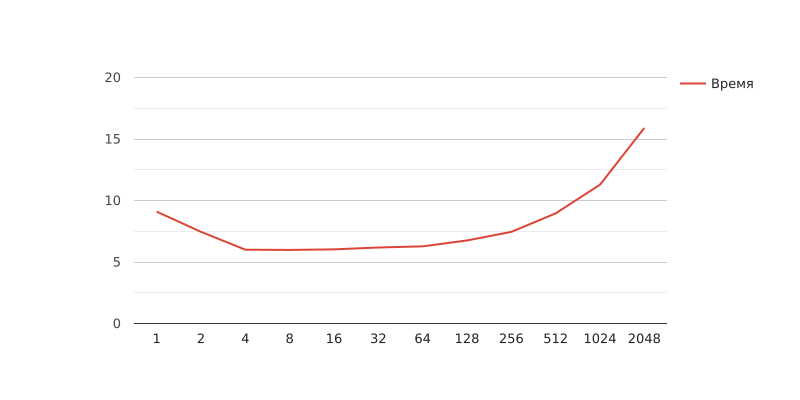
}

**Пример работы**

Многопоточные программы работают быстрее обычных, если их хорошо распараллелить.



Однако, если использовать слишком много потоков, программа будет работать медленнее.



**Тесты:**

**1.**

1 0 0

0 0 0

10 0 0

5 0 0

0 5 0

0 15 0

0 0 0

0 1 0

0 4 0

0 0 0

Максимальная площадь треугольника: 75

***Вывод программы:***

Count threads: 5

Count points: 10

1 0 0

0 0 0

10 0 0

5 0 0

0 5 0

0 15 0

0 0 0

0 1 0

0 4 0

0 0 0

Start!

Итоговое время: 8 миллисекунд

Итоговый максимум: 75

**2.**

10 0 10

14 0 0

5 0 0

8 0 9

1 0 4

Максимальная площадь треугольника: 57

***Вывод программы:***

Count threads: 8

Count points: 5

10 0 10

14 0 0

5 0 0

8 0 9

1 0 4

Start!

Итоговое время: 10 миллисекунд

Итоговый максимум: 57

**3.**

10 0 0

0 0 10

0 0 0

10 0 0

0 0 10

0 0 0

10 0 0

0 0 10

0 0 0

10 0 0

0 0 10

...

(и так 510 строчек)

...

0 0 10

0 0 0

10 0 0

0 0 10

0 0 0

10 0 0

0 0 10

0 0 0

Максимальная площадь треугольника: 50

***Вывод программы:***

Count threads: 1

Count points: 510

10 0 0

0 0 10

0 0 0

10 0 0

0 0 10

0 0 0

10 0 0

0 0 10

0 0 0

...

(и так 510 строчек)

...

0 0 0

10 0 0

0 0 10

0 0 0

10 0 0

0 0 10

0 0 0

Start!

Итоговое время: 4906 миллисекунд

Итоговый максимум: 50

**4.**

**При 1 потоке**

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

10 0 0

0 10 0

5 0 0

0 0 5

0 0 0

0 0 0

0 0 0

Максимальная площадь треугольника: 61.2372

***Вывод программы:***

Count threads: 1

Count points: 15

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

10 0 0

0 10 0

5 0 0

0 0 5

0 0 0

0 0 0

0 0 0

Start!

Итоговое время: 1 миллисекунд

Итоговый максимум: 61.2372

**5.**

**При 6 потоках**

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

10 0 0

0 10 0

5 0 0

0 0 5

0 0 0

0 0 0

0 0 0

Максимальная площадь треугольника: 61.2372

***Вывод программы:***

Count threads: 6

Count points: 15

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

10 0 0

0 10 0

5 0 0

0 0 5

0 0 0

0 0 0

0 0 0

Start!

Итоговое время: 2 миллисекунд

Итоговый максимум: 61.2372

**Вывод**

Выполнив эту задачу, я смог успешно реализовать разделение вычислений на множество параллельных процессов, что позволило эффективнее использовать доступные ресурсы и значительно ускорить поиск максимального значения площади путем перебора всех потенциальных точек x, y и z. Помимо этого, я также создал возможность для пользователя самому выбирать количество используемых потоков.

У меня возникли трудности когда я потокам передал один и тот же класс. Когда потоки работают паралельно с одними и теми же данными они друг у друга меняли значения, что вело к ошибкам при работе с несколькими потоками. Решил эту проблему путем удаления класса и созданием отдельной структуры для каждого потока.

В результате, я приобрел практический опыт работы с многозадачностью и параллельными вычислениями в операционной системе UNIX.

На основе построенных графиков можно сделать вывод, что использование 10 потоков является оптимальным.