Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 3 по курсу**

**«Операционные системы»**

**Управление потоками в ОС**

Студент: Бутырев Д.В.

Группа: М8О–206Б–20

Вариант: 12

Преподаватель: Соколов Андрей Алексеевич

Оценка: 5

Дата: 23.10.21

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Постановка задачи**

## Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков 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 входящих данных и количества потоков. Получившиеся результаты необходимо объяснить.

Вариант 12: Наложить K раз фильтр, использующий матрицу свертки, на матрицу, состоящую из вещественных чисел. Размер окна 3х3

**Общие сведения о программе**

Программа использует потоки, для этого вызываются функции:

1. **pthread\_create ––** (является оберткой над системным вызовом clone) создает новый поток в вызывающем процессе. В качестве аргументов принимает указатель на структуру-идентификатор потока pthread\_t, атрибуты потока, функцию, которая будет запускаться в потоке, список аргументов для функции в виде указателя на void. В случае успеха возвращает 0, иначе возвращает номер ошибки.
2. **pthread\_join ––** используется для ожидания завершения потока. Данная функция блокирует вызывающий поток, пока указанный поток не завершится. В качестве аргументов принимает структуру pthread\_t потока и указатель на переменную, в которую будет записан результат, возвращаемый потоком. В случае успеха возвращает 0, иначе возвращает номер ошибки.

**Общий метод и алгоритм решения**.

1. Изучить реализацию threads в С,

2. Разработать алгоритм распараллеливания задачи,

3. Написать программу

**Основные файлы программы**

#include <utility>

#include <vector>

using matrix = std::vector<std::vector<double>>;

struct PthreadData {

PthreadData(matrix& pixel\_matr, matrix& copy\_pixel\_matr, matrix& convolution\_matr, std::pair<int, int>& current\_pixel)

: pixel\_matr(pixel\_matr), copy\_pixel\_matr(copy\_pixel\_matr), convolution\_matr(convolution\_matr), current\_pixel(current\_pixel), \

pixel\_matr\_height(pixel\_matr.size()), pixel\_matr\_width(pixel\_matr[0].size()),\

convolution\_matr\_side(convolution\_matr.size()) {}

matrix& pixel\_matr;

matrix& copy\_pixel\_matr;

matrix& convolution\_matr;

std::pair<int, int> current\_pixel;

const int pixel\_matr\_height; //can't use &&, why?

const int pixel\_matr\_width;

const int convolution\_matr\_side;

void calculate() {

double sum(0);

std::pair<int, int> cur = std::pair<int,int>(current\_pixel.first-convolution\_matr\_side/2, \

current\_pixel.second-convolution\_matr\_side/2);

for(int cur\_conv\_y = 0, cur\_pixel\_y = cur.first; cur\_conv\_y < convolution\_matr\_side; cur\_conv\_y++, cur\_pixel\_y++) {

for(int cur\_conv\_x = 0, cur\_pixel\_x = cur.second; cur\_conv\_x < convolution\_matr\_side; cur\_conv\_x++, cur\_pixel\_x++) {

int koef\_x(0), koef\_y(0);

if(cur\_pixel\_x < 0) koef\_x = -cur\_pixel\_x;

else if(cur\_pixel\_x > static\_cast<int>(pixel\_matr\_width-1)) koef\_x = -cur\_pixel\_x + pixel\_matr\_width-1;

if(cur\_pixel\_y < 0) koef\_y = -cur\_pixel\_y;

else if(cur\_pixel\_y > static\_cast<int>(pixel\_matr\_height-1)) koef\_y = -cur\_pixel\_y + pixel\_matr\_height-1;

sum += pixel\_matr[cur\_pixel\_y + koef\_y][cur\_pixel\_x + koef\_x] \* convolution\_matr[cur\_conv\_y][cur\_conv\_x];

}

}

copy\_pixel\_matr[current\_pixel.first][current\_pixel.second] = sum;

}

};

//#include <unistd.h>

#include <iostream>

#include <sys/types.h>

//#include <cstdlib>

#include <pthread.h>

#include <vector>

#include <utility>

#include "PthreadData.cpp"

#include <vector>

#include <ctime>

void\* thread\_func(void\* data) {

static\_cast<PthreadData\*>(data)->calculate();

return NULL;

}

void print\_matr(matrix& m) {

for(unsigned int i = 0; i < m.size(); i++) {

for(unsigned int j = 0; j < m[0].size(); j++)

std::cout << m[i][j] << '\t';

std::cout << std::endl;

}

}

using matrix = std::vector<std::vector<double>>;

int main() {

unsigned int start\_time = clock();

unsigned int num\_of\_threads;

unsigned int pixel\_matr\_height, pixel\_matr\_width;

unsigned int convolution\_matr\_side; // %2=1

//double\* pixel\_matr = new double [pixel\_matr\_height\*pixel\_matr\_width];

//double copy\_pixel\_matr[pixel\_matr\_height][pixel\_matr\_width] = { 0 };

//double convolution\_matr[convolution\_matr\_side][convolution\_matr\_side];

std::cin >> pixel\_matr\_height >> pixel\_matr\_width;

matrix pixel\_matr(pixel\_matr\_height, std::vector<double>(pixel\_matr\_width, 0));

matrix copy\_pixel\_matr = pixel\_matr; //strange, when I put this declaration above size input, there where no compile-errors, but abortion

//before the start of the prog

for(unsigned int i = 0; i < pixel\_matr\_height; i++)

for(unsigned int j = 0; j < pixel\_matr\_width; j++)

std::cin >> pixel\_matr[i][j];

std::cin >> convolution\_matr\_side;

matrix convolution\_matr(convolution\_matr\_side, std::vector<double>(convolution\_matr\_side, 0));

for(unsigned int i = 0; i < convolution\_matr\_side; i++)

for(unsigned int j = 0; j < convolution\_matr\_side; j++)

std::cin >> convolution\_matr[i][j];

std::cin >> num\_of\_threads;

std::pair<int,int> init\_current\_pixel(std::pair<int,int>(0,0));

PthreadData init\_unit(pixel\_matr, copy\_pixel\_matr, convolution\_matr, init\_current\_pixel);

std::vector<pthread\_t> threads(num\_of\_threads); //arr i[b] -- intresting, this array in dynamic memory, should I free() it? why I can make it?

std::vector<PthreadData> pthreaddata(num\_of\_threads, init\_unit);

const unsigned int&& num\_of\_pixels = pixel\_matr\_height \* pixel\_matr\_width;

if(num\_of\_threads > num\_of\_pixels) num\_of\_threads = num\_of\_pixels;

const unsigned int num\_of\_execution = 3;

/\*std::cout << "original:" << std::endl;

print\_matr(pixel\_matr);

std::cout << "copy:" << std::endl;

print\_matr(copy\_pixel\_matr);\*/

for(unsigned int i = 0; i < num\_of\_execution; i++) {

unsigned int pixels\_done(0); //i don't undersrtand how does it work? initialising in loop

while(pixels\_done < num\_of\_pixels) {

unsigned int threads\_to\_open = std::min(num\_of\_threads, num\_of\_pixels-pixels\_done);

for(unsigned int j = 0; j < threads\_to\_open; j++) {

int&& current\_pixel = (pixels\_done + 1) - 1;

pthreaddata[j].current\_pixel = std::pair<int,int>(current\_pixel/pixel\_matr\_width, current\_pixel%pixel\_matr\_width);

++pixels\_done;

pthread\_create(&(threads[j]), NULL, thread\_func, &(pthreaddata[j]));

}

for(unsigned int j = 0; j < threads\_to\_open; j++) {

pthread\_join(threads[j], NULL); // if they will complete unconsistanly, and I will wait for 1, but everyone

//has already finished?

}

/\*std::cout << "original:" << std::endl;

print\_matr(pixel\_matr);

std::cout << "copy:" << std::endl;

print\_matr(copy\_pixel\_matr);\*/

}

/\*for(unsigned int j = 0; j < num\_of\_threads; j++) {

std::swap(pthreaddata[j].pixel\_matr, pthreaddata[j].copy\_pixel\_matr);

}\*/

std::swap(pixel\_matr, copy\_pixel\_matr); //if I swap in main, will objects, that have this units as reference seap?

pixels\_done = 0;

}

/\*matrix\* output\_matr(NULL); //can I do that with & or &&?

if(num\_of\_execution%2 == 0) output\_matr = &pixel\_matr;

else output\_matr = &copy\_pixel\_matr;

print\_matr(\*output\_matr);\*/

//print\_matr(pixel\_matr);

unsigned int end\_time = clock();

std::cout << end\_time - start\_time << std::endl;

}

**Пример работы**

steep@gg:~/educat/instit/os/3lab/src/build$ ./os\_lab\_03

2 2

1 1

1 1

1

4

2

64 64

64 64

1809

steep@gg:~/educat/instit/os/3lab/src/build$ ./os\_lab\_03

3 3

1 1 1

1 1 1

1 1 1

1

3

4

27 27 27

27 27 27

27 27 27

2228

**Вывод**

Использование потоков гораздо проще, чем использовать процессы для распараллеливания программы. Для этого существует простой и понятный интерфейс. При использовании потоков могут возникать неприятные проблемы с памятью, но их можно избежать, используя различные методы синхронизации и составляя безопасные алгоритмы.

Average result for 0 test

1 thread: 1925.1000000000

2 thread: 1475.9000000000

3 thread: 1504.6000000000

4 thread: 1211.9000000000

5 thread: 1313.8000000000

6 thread: 1287.3000000000

7 thread: 1258.7000000000

8 thread: 1230.4000000000

9 thread: 1361.7000000000

Average result for 1 test

1 thread: 6666.4000000000

2 thread: 4572.4000000000

3 thread: 3592.5000000000

4 thread: 3709.5000000000

5 thread: 3971.4000000000

6 thread: 4279.2000000000

7 thread: 5036.0000000000

8 thread: 5168.5000000000

9 thread: 4484.3000000000

Average result for 2 test

1 thread: 15307.8000000000

2 thread: 10415.0000000000

3 thread: 8874.2000000000

4 thread: 7949.2000000000

5 thread: 8544.2000000000

6 thread: 10148.4000000000

7 thread: 10157.5000000000

8 thread: 9556.4000000000

9 thread: 10742.5000000000

Average result for 3 test

1 thread: 4176327.5000000000

2 thread: 2898728.1000000000

3 thread: 2284220.4000000000

4 thread: 2274407.9000000000

5 thread: 2431727.2000000000

6 thread: 2692067.8000000000

7 thread: 2741860.8000000000

8 thread: 3186852.5000000000

9 thread: 3475188.0000000000

В среднем наибольшая скорость программы достигается на 4 потоках. При дальнейшем увеличении потоков происходит увеличение времени программы — это может происходить по следующим причинам: количеством ядер ограниченно, при увеличении числа потоков увеличивается количество системных вызовов, параллелится только часть программы.