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: Волков Евгений Андреевич
Группа: М8О–207Б–21
Вариант: 12
Преподаватель: Миронов Евгений Сергеевич

Москва, 2023

**Репозиторий**

## 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 3x3

## Общие сведения о программе

Программа написана на языке C++, используя библиотеку <pthread.h>.

**Общий метод и алгоритм решения**.

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.

Вводим исходную матрицу, матрицу свертки размера 3, количество потоков, количество применений матрицы k. Затем создаем вектор потоков, вектор классов, накладывающих матрицу свертки на 1 пиксель. Далее вычисляем количество элементов матрицы, распараллеливаем пиксели по потокам, создаем потоки, которые будут параллельно вызывать метод класса, накладывающий фильтр на 1 пиксель и ждем завершения их работы, пока не пробежимся по всем пикселям. Затем повторим операцию k раз.

## Исходный код

main.cpp

```
#include <iostream>
#include <pthread.h>
#include <vector>
#include <ctime>

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

struct PthreadData
{
    matrix &pixel_matr;         // матрица
    matrix &copy_pixel_matr;    // ее копия (результирующая матрица)
    matrix &convolution_matr;   // матрица свертки
    std::pair<int, int> current_pixel;  //  пиксель (скорее всего i, j элемента матрицы)
    int pixel_matr_height;        // размеры матрицы
    int pixel_matr_width;
    int convolution_matr_side;    // размер матрицы свертки
    PthreadData(matrix &pixel_matr, matrix &copy_pixel_matr, matrix &convolution_matr, std::pair<int, int> &current_pixel); // констуктор
    void calculate();             // функция подсчета свертки
};

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())
{
```

```
}


void PthreadData::calculate()   // обрабатывает 1 элемент
{
   double sum(0);  // сумма
   std::pair<int, int> cur = std::pair<int, int>(current_pixel.first - convolution_matr_side / 2,
                                                 current_pixel.second - convolution_matr_side / 2); // -1, -1
   // идем переменными cur_conv_y и cur_conv_x по матрице свертки, cur_pixel_y а cur_pixel_y и cur_pixel_x по исходной матрице (не полностью)
   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 > (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 > (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;
}


void *thread_func(void *data)       // функция потока (рутина), которая накладывает фильтр на матрицу
{
```

```
    static_cast<PthreadData *>(data)->calculate(); // static_cast преобразует void * в класс
    return NULL;
}

void print_matr(matrix &m)
{
    for (int i = 0; i < m.size(); i++)
    {
        for (int j = 0; j < m[0].size(); j++)
        {
            std::cout << m[i][j] << '\t';
        }
        std::cout << std::endl;
    }
}

int main(int argc, char const *argv[])
{
    int start_time = clock(); // начало замера времени
    int num_of_threads;       // количество потоков
    int pixel_matr_height, pixel_matr_width; // размеры матрицы
    int convolution_matr_side = 3;           // размеры матрицы свертки (размера 3 на 3, по условию)

    std::cout << "Enter the height and width of matrix\n";
    std::cin >> pixel_matr_height >> pixel_matr_width;

    matrix pixel_matr(pixel_matr_height, std::vector<double>(pixel_matr_width, 0)); // создаем матрицу нужных размеров
    matrix copy_pixel_matr = pixel_matr;                                            // копия матрицы

    std::cout << "Enter the matrix\n";
    for (int i = 0; i < pixel_matr_height; i++)                                     // заполняем матрицу
    {
        for (int j = 0; j < pixel_matr_width; j++)
        {
            std::cin >> pixel_matr[i][j];
        }
    }
```

```
    matrix convolution_matr(convolution_matr_side, std::vector<double>(convolution_matr_side, 0)); // создаем матрицу свертки 3 на 3

    std::cout << "Enter the convolution matrix size of 3x3\n";
    for (int i = 0; i < convolution_matr_side; i++)                                    // заполняем матрицу свертки
    {
        for (int j = 0; j < convolution_matr_side; j++)
        {
            std::cin >> convolution_matr[i][j];
        }
    }

    std::cout << "Enter number of threads\n";
    std::cin >> num_of_threads;

    std::pair<int, int> init_current_pixel(std::pair<int, int>(0, 0));              //  пиксель (скорее всего i, j элемента матрицы)
    PthreadData init_unit(pixel_matr, copy_pixel_matr, convolution_matr, init_current_pixel);   // создаем объект класса, применяющего алгоритм свертки к матрице
    std::vector<pthread_t> threads(num_of_threads);                                    // вектор потоков
    std::vector<PthreadData> pthreaddata(num_of_threads, init_unit);                   // вектор классов для работы над матрицами для каждого потока

    int &&num_of_pixels = pixel_matr_height * pixel_matr_width;                        // количество элементов матрицы
    if (num_of_threads > num_of_pixels)
    {
        num_of_threads = num_of_pixels;
    }

    int num_of_execution;

    std::cout << "Enter k\n";
    std::cin >> num_of_execution;                                                      // количество применений алгоритма

    for (int i = 0; i < num_of_execution; i++)
    {
        int pixels_done = 0;
        while (pixels_done < num_of_pixels)                                            // для каждого элемента применяем алгоритм
        {
            int threads_to_open = std::min(num_of_threads, num_of_pixels - pixels_done); // количество необходимых потоков для обработки num_of_pixels элементов
```

```
        for (int j = 0; j < threads_to_open; j++)                    // за одну итерацию можем параллельно обработать threads_to_open элементов
        {
            int &&current_pixel = (pixels_done + 1) - 1;                    // вычисляем текущий элемент
            // присваиваем текущему элементу класса j-го потока координаты current_pixel в матрице
            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])); // создает поток threads[j], который будет работать с аргументами pthreaddata[j]
        }
        for (int j = 0; j < threads_to_open; j++)
        {
            pthread_join(threads[j], NULL); // ждет поток и вывод потока
        }
      }
      std::swap(pixel_matr, copy_pixel_matr); // присваиваем результат исходной матрице
      pixels_done = 0;
   }
   std::cout << "Result matrix:\n";
   print_matr(pixel_matr);
   int end_time = clock(); // время в конце
   std::cout << "Time:\n";
   std::cout << end_time - start_time << std::endl;
}
```

# Демонстрация работы программы

```
ggame@ggame:~/hubs/newos/evgeny/OC-labs/lab3/build$ ./main
Enter the height and width of matrix
7 7
Enter the matrix
1 2 3 4 5 6 7
2 5 4 7 8 9 1
3 5 2 4 7 8 9
4 1 0 2 5 7 8
5 6 2 1 4 7 8
6 1 2 3 4 5 7
```

7 5 2 1 4 5 6

Enter the convolution matrix size of 3x3

1 1 1

1 1 1

1 1 1

Enter number of threads

3

Enter k

3

Result matrix:

| | | | | | | |
|---|---|---|---|---|---|---|
| 1904 | 2226 | 2827 | 3510 | 4093 | 4382 | 4467 |
| 2068 | 2293 | 2783 | 3433 | 4084 | 4491 | 4662 |
| 2284 | 2345 | 2637 | 3212 | 3953 | 4548 | 4862 |
| 2553 | 2421 | 2453 | 2884 | 3667 | 4451 | 4926 |
| 2824 | 2529 | 2323 | 2582 | 3311 | 4160 | 4708 |
| 3142 | 2729 | 2334 | 2421 | 3033 | 3846 | 4392 |
| 3342 | 2872 | 2382 | 2370 | 2888 | 3633 | 4140 |

Time:

6088

## 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.