Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу
«Операционные системы»

Тема работы
“Потоки”

Студент: Рылов Александр Дмитриевич
Группа: М8О-207Б-21
Вариант: 12
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

Репозиторий

https://github.com/artemmoroz0v/OS

Постановка задачи

Наложить K раз фильтр, использующий матрицу свертки, на матрицу, состоящую из вещественных чисел. Размер окна 3x3

Общие сведения о программе

Для реализации поставленной задачи нам нужны следующие библиотеки:

<stdlib.h> - для функций, работающими с памятью

<time.h> - для функций, работающих со временем

<pthread.h> - для работы с потоками.

<string.h> - для использования функций над строками.

<errno.h> - для вывода ошибок

Программа собирается и запускается при помощи следующих команд:

gcc main.c -pthread -o main

./main convolutions_number thread_nuber (пример: ./main 3 9).

Общий метод и алгоритм решения

Считывается количество применений матрицы свертки, количество тредов, матрица изображения. Далее запускается некоторое количество потоков, не более заданного. Из-за размеров матрицы свертки (3x3), эффективно работать могут до 9 тредов (по одному на каждый элемент матрицы). Преимущество использования потоков в скорости исполнения будет заметно при размере обрабатываемого изображения >= 1000x1000 — до этого быстрее работает однопоточный вариант из-за затрат на инициализацию тредов.

Исходный код

```
void *convolution(void *a) {
    Thread_args *args = (Thread_args *)a;

    int n_elems = CONV_SIZE * CONV_SIZE; //всего элементов в матрице
    int n_opers = n_elems / args->thread_cnt; //сколько операций на поток
    int n_rest = n_elems % args->thread_cnt; //остаток операций

    int start, end;
    if (args->thread_number == 0) {
        start = args->thread_number * n_opers;
        end = (args->thread_number + 1) * n_opers + n_rest; //первый поток делает больше
    } else {
        start = args->thread_number * n_opers + n_rest;
        end = (args->thread_number + 1) * n_opers + n_rest;
    }

    int row_offset = 0, col_offset = 0;
    while (CONV_SIZE + row_offset <= args->new_im_size) {
        col_offset = 0;
        while (CONV_SIZE + col_offset <= args->new_im_size) {
            for (int op = start; op < end; ++op) {
                int row = op / CONV_SIZE;
                int col = op % CONV_SIZE;

                args->res_matrix[get_ind(row_offset, col_offset, args->res_size)] += conv_matrix[row][col] *
                args->new_img_matrix[get_ind(row_offset + row, col_offset + col, args->new_im_size)];
            }
        ++col_offset;
        }
    ++row_offset;
    }
}
```

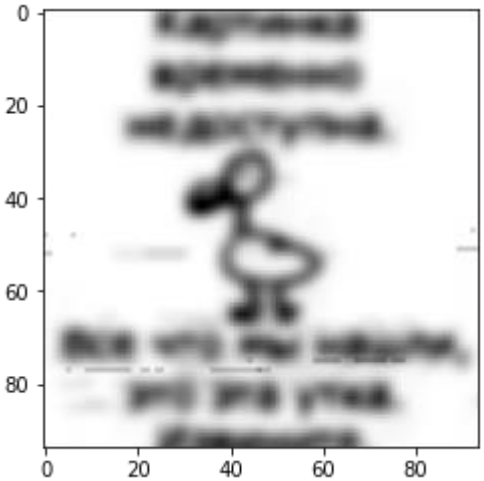
```
pthread_t *threads = (pthread_t *) malloc(sizeof(pthread_t) * thread_cnt);
Thread_args *th_args = (Thread_args *) malloc(sizeof(Thread_args) * thread_cnt);

for (int i = 0; i < conv_cnt; ++i) { //применить k раз свертку
    if (i != 0) {
        new_im_size = res_size;
        new_img_matrix = (double*) realloc(new_img_matrix ,sizeof(double) * new_im_size * new_im_size);
        memcpy(new_img_matrix, res_matrix, sizeof(double) * new_im_size * new_im_size);
    }

    res_size = new_im_size - CONV_SIZE + 1;
    if (res_size < 0) { //матрицу свертки можно применить только к матрице размером >= 3x3
        res_size = 1;
        break;
    }
    res_matrix = (double *) realloc(res_matrix, sizeof(double) * res_size * res_size);
    memset(res_matrix, 0, sizeof(res_matrix) * res_size * res_size);

    thread_args_init(th_args, conv_cnt, thread_cnt, new_im_size, res_size, new_img_matrix, res_matrix);

    for (int j = 0; j < thread_cnt; ++j) {
        if (pthread_create(&threads[j], NULL, convolution, (void *) &th_args[j]) != 0) {
            perror("Failed to create thread");
            return 3;
        }
    }

    for (int j = 0; j < thread_cnt; ++j) {
        if (pthread_join(threads[j], NULL) != 0) {
            perror("Failed to join thread");
            return 3;
        }
    }

    //printf("intermediate res:\n");
    //print(res_size, res_matrix);
}
```

Демонстрация работы программы

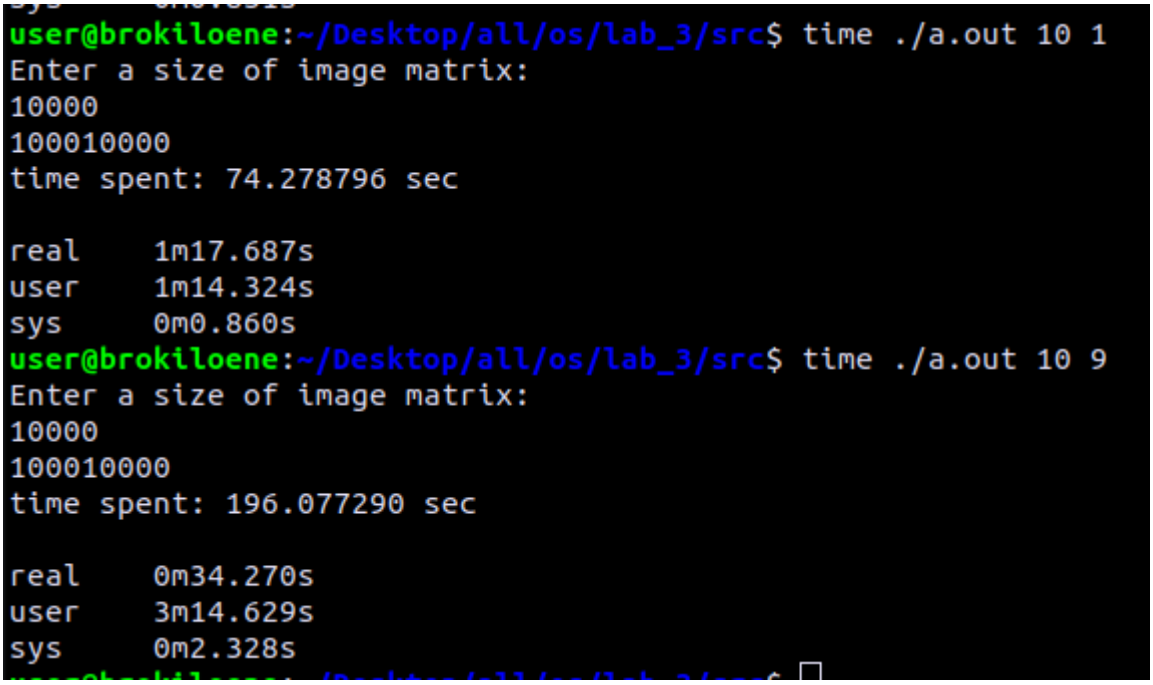
Программа выводит обработанную матрицу изображения, лучше показать результат в картинках.

Изображение до обработки

После обработки



Выигрыш по времени:



Для матрицы 10000x10000 — более чем в два раза (смотреть по real, при первом запуске использовался один поток, во втором запуске 9)

Выводы

Данная лабораторная работа помогла мне успешно ознакомиться с тем, как устроены потоки в Linux. Во время выполнения своего задания я изучил особенности системных вызовов и узнал многие тонкости работы с потоками.