Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект
по курсу «Операционные системы»
III Семестр

Изучение многопоточности в задаче редактирования растрового изображения

Студент:	Синявский А.В
Группа:	М80-208Б-18
Преподаватель:	Миронов Е.С
Оценка:	
Дата:	

Москва 2019

Содержание

1. Цель проекта

Цель моего проекта – написать программу на языке C++, которая бы накладывала на изображение в формате BMP определённые графические фильтры, а именно инверсию цветов и чёрно-белый фильтр (точнее фильтр, переводящий все цвета в оттенки серого). Программа должна использовать многопоточность для решения этой задачи. Также необходимо проанализировать скорость работы программы при различном числе потоков, вывести оптимальное число, а также постараться теоретически обосновать полученные результаты.

2.Программа

2.1 Немного о формате BMP

Устройство данного формата в общих чертах довольно просто: в начале файла находится различная информация о файле, такая как ключ самого формата, размер файла, разрешение изображения, колличество бит, которыми кодируется один пиксель и так далее. После всей заголовочной информации идут данные о самих пикселях. Для программы нам нужна информация о размере файла, размере матрицы пикселей, позиции в файле, с которой начинаются пиксели, а также то, сколько бит уходит на один пиксель.

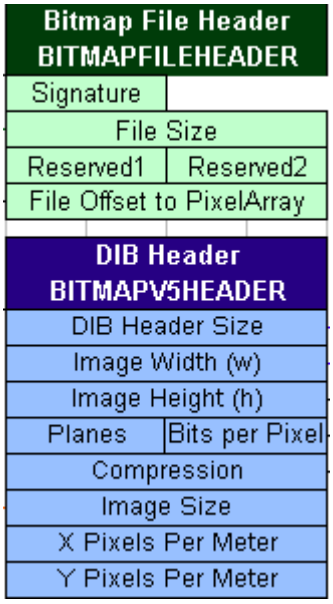


Иллюстрация 1: Часть структуры .bmp файла

2.2. Идея реализации программы

Задумка заключается в том, чтобы сначала без использования многопоточности считать из хэдера файла всю необходимую информацию, затем отобразить весь файл в память при помощи функции mmap, и запустить N потоков, каждому из которых во владение будет предоставлен свой кусок памяти, на которую отображён файл. Так как основная задача проета – найти оптимальное число потоков, для простоты реализации уместно будет выделять потокам пиксели построчно.

2.3. Код

```
#include <iostream>
#include <fstream>
#include <fcntl.h>
#include <sys/mman.h>
#include <cstdlib>
#include <thread>
#include <condition_variable>
#include <mutex>

std::mutex mutex;
std::condition_variable cv;

void reverse(unsigned int &start, unsigned int &end, unsigned char* mapping) {
    mutex.lock();
    int st = start, nd = end;
    cv.notify_all();
    mutex.unlock();
    int tmp;

    for (int i = st; i < nd; i++) {
        tmp = (int) mapping[i];
        mapping[i] = char(255 - tmp);
    }
}

void black_n_white(unsigned int &start, unsigned int &end, unsigned char* mapping) {
    mutex.lock();
    int st = start, nd = end;
    cv.notify_all();
    mutex.unlock();
    unsigned char tmp = 0;

    for (int i = st; i < nd; i+=3) {
```

```
        tmp = (unsigned char)((mapping[i] + mapping[i+1] + mapping[i+2])/3);

        mapping[i] = tmp;
        mapping[i+1] = tmp;
        mapping[i+2] = tmp;
    }
}

int main(int argc, char* argv[]) {
    if (argc < 2) {
        std::cout << "USAGE: ./main <file.pmp> <number of threads> <filter option>\n";
        return -1;
    }

    //=========================preparations=============================//

    std::ifstream file(argv[1], std::ios::binary);
    if (file.bad()) {
        std::cout << "Bad File\n";
        return -1;
    }
    u_short tmp2, depth;
    uint32_t tmp4, size, m_offset, width, height;
    file.read((char*)&tmp2, sizeof(tmp2));
    file.read((char*)&size, sizeof(size));
    file.read((char*)&tmp4, sizeof(tmp4));
    file.read((char*)&m_offset, sizeof(m_offset));
    file.read((char*)&tmp4, sizeof(tmp4));
    file.read((char*)&width, sizeof(width));
    file.read((char*)&height, sizeof(height));
    file.read((char*)&tmp2, sizeof(tmp2));
    file.read((char*)&depth, sizeof(depth));
    if (depth != 24) {
        std::cout << "unfortunately, this program doesn`t know how to process .bmp files \n"
                  "with each pixel coded NOT by 24 bits (byte per colour in RGB term)\n"
                  "please contact developer and say him that somehow your file has weird color coding/\
n";
        return -1;
    }
    std::cout << "size of picture: " << width << 'x' << height << '\n';
    file.close();

    //=========================preparations-done=========================//

    int fd = open(argv[1], O_RDWR);
    if (fd < 0) {
        std::cout << "can`t open file\n";
        return -1;
    }
    uint32_t n_of_thrs;
```

```
    int filter_type;
    n_of_thrs = atoi(argv[2]);
    filter_type = atoi(argv[3]);
    if (n_of_thrs > height) n_of_thrs = height;
    std::thread threads[n_of_thrs];
    unsigned char* mapping = (unsigned char*)(mmap(nullptr, size, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0));
    if (mapping == MAP_FAILED) {
        std::cout << "mmap failed\n";
        return -1;
    }
    uint32_t pool = (height / n_of_thrs) * width * 3;
    uint32_t leftover = (height % n_of_thrs) * width * 3;
    uint32_t start = m_offset;
    uint32_t end = m_offset + pool;

    std::unique_lock<std::mutex> lock(mutex);
    if (filter_type == 1) {
        for (int i = 0; i < n_of_thrs; i++) {
            threads[i] = std::thread(reverse, std::ref(start), std::ref(end), mapping);
            cv.wait(lock);
            start += pool;
            if (i == n_of_thrs - 2) {
                end += pool + leftover;
            } else
                end += pool;
        }
    } else if (filter_type == 2) {
        for (int i = 0; i < n_of_thrs; i++) {
            threads[i] = std::thread(black_n_white, std::ref(start), std::ref(end), mapping);
            cv.wait(lock);
            start += pool;
            if (i == n_of_thrs - 2) {
                end += pool + leftover;
            } else
                end += pool;
        }
    }
    mutex.unlock();

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

2.4. Пояснения к коду

Первая часть функции main, помеченная как preparations, служит для поиска в заголовке файла всей нужной информации. Программа рассчитана только на кодировку пикселя 24 битами, поэтому программа выдаёт ошибку, если этого не происходит. После идёт основная часть, в которой задаётся отобраение файла в память и запускаются потоки. Потоки создаются по следующему принципу: каждому потоку выделяется определённое число строк, по возможности равное у всех потоков. Если строки на необходимое число потоков нацело не делятся, то остаток от деления скидывается на обработку последнему потоку. Следует отметить, что создание потоков происходит строго последовательно, цикл каждый раз ждёт сигнала от потока после его запуска, чтобы в 2 разных потока не попала одна и та же область файла.

3. Пример работы программы

Чтобы проиллюстрировать работу программы, приведу 2 скриншота в стиле “до/после обработки”. Сама картинка имеет разрешение 7680x4320.

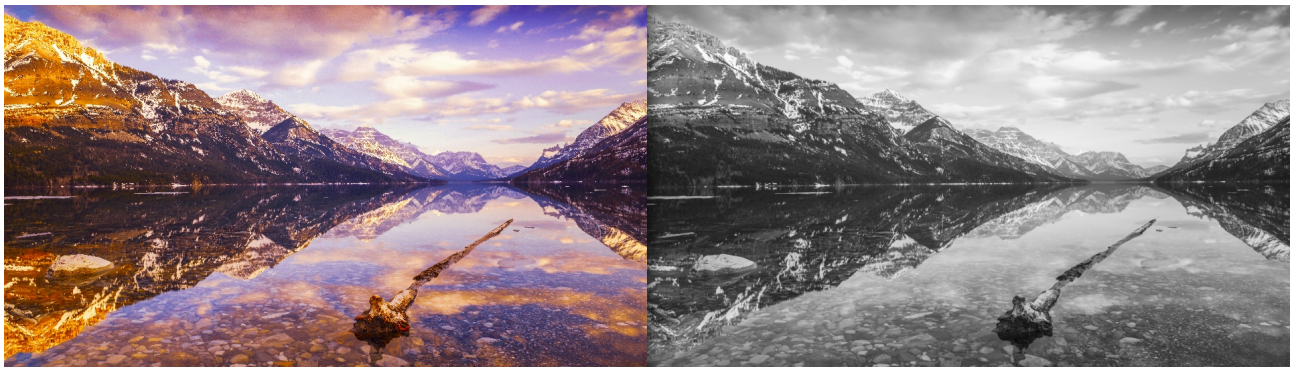


Иллюстрация 2: Применение чёрно-белого фильтра

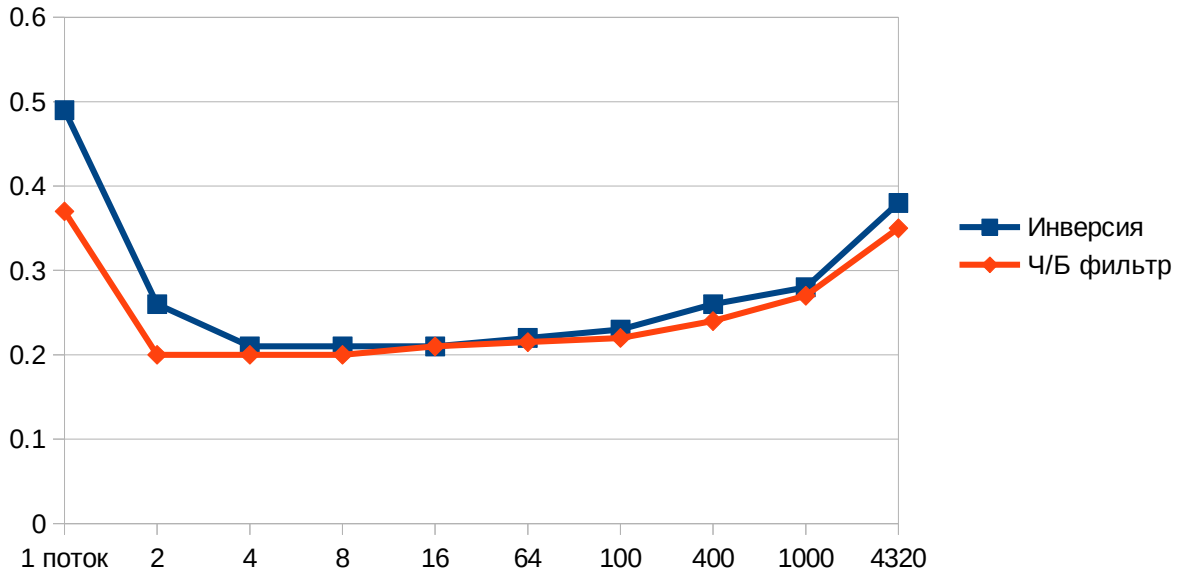
Иллюстрация 3: Применение инверсии цветов

4. Анализ

4.1. График зависимости

Так как все условия программе задаются при её запуске в командной строке, можно без проблем замерить время её работы при помощи утилиты time. Она выводит реальное, системное и пользовательское время работы программы. Так как системное и пользовательское время учитывают не то, сколько фактически работала программа, а число тактов процессора, поделённое на число тактов в секунду, что не самым очевидным образом взаимодействует с многопоточностью, нам будет интересно именно реальное время работы программы.

Иллюстрация 4: График зависимостей



4.2. Анализ графика

По этому приблизительному графику можно сделать вывод, что оптимальное число потоков для решения задачи – 4. Но нам не нужно просто число, нам нужны ответы! Понять, почему график выглядит именно так, нам поможет закон Амдала. Этот закон говорит о следующем: в случае, когда задача разделяется на несколько частей, суммарное время её выполнения на параллельной системе не может быть меньше времени выполнения самого

медленного фрагмента, при условии одинаковой скорости всех вычислителей. Значит, время работы программы “ограничено снизу” самой длинной частью кода, которую нельзя распараллелить. На первый взляд, этой частью может показаться подготовительная секция, в которой программа получает из файла нужную информацию. Но на самом деле, интереснее другой сегмент кода, а именно цикл, в котором создаются потоки. Так как каждому потоку в распоряжение должен попасть строго определённый кусок отображаемого файла, потоки должны создаваться строго последовательно. Поток 1 создался, основной поток убедился, что поток 1 получил свой кусок файла, основной поток обновил данные о выделяемом куске файла, передал их в качестве входных данных потоку 2, начал ждать сигнала об успешном старте от потока 2… и так пока не запустятся все потоки. Этот цикл невозможно распараллелить, и он становится тем больше, чем больше потоков мы будем запускать. Поэтому в какой-то момент время работы программы начинает увеличиваться, и график напоминает параболу.

Ещё один вопрос – почему разницы между 4 и 16 потоками практически нет? Для ответа на этот вопрос нужно разобраться в том, как вообще работает многопоточность. Вообще, потоки далеко не всегда и не все работают одновременно. Зачастую они работают, поочерёдно сменяя друг друга. Это происходит из-за того, что процессор может одновременно выполнять строго ограниченное число задач. Если у процессора 2 ядра, соответственно и задач 2. Поэтому из 16 потоков параллельно работают только 2 (в моём случае на самом деле 4), а остальные ждут своей очереди, из-за чего время не меняется. На самом деле оно даже ухудшается из-за затрат на переключение между активными потоками, но в данном случае эта разница составляет тысячные доли секунды, и её довольно сложно проследить. Так почему 4, а не 2? В процессоре на моём компьютере 2 ядра, но он оснащён технологией Hyper-Threading Technology (HTT), суть которой заключается в том, что одно физическое ядро процессора представляется как 2 логических.

Вывод

Благодаря данному проекту я узнал много нового о цветовых фильтрах и алгоритмах их работы, об устройстве графических файлов, вспомнил что тип данных char в C++ знаковый, и впервые столкнулся с задачей, где это имело значение, улучшил своё понимание принципов работы многопоточных программ, больше узнал о работе процессора, а также улучшил свои навыки поиска информации и анализа.