Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Межпоточное взаимодействие**

Студент: Шараковский Юрий Дмитриевич

Группа: М8О–206Б–20

Вариант: 21

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**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 обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). При создании необходимо предусмотреть ключи, которые позволяли бы задать максимальное количество потоков, используемое программой. При возможности необходимо использовать максимальное количество возможных потоков. Ограничение потоков может быть задано или ключом запуска вашей программы, или алгоритмом.

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

Вариант 10: Наложить K раз медианный фильтр на матрицу состоящую из целых чисел. Размер окна задается.

**Общие сведения о программе**

Программа компилируется из файла main.c. Также используется заголовочные файлы: unistd.h, fcntl.h, sys/types.h, sys/stat.h, sys/syscall.h, sys/time.h, stdio.h, stdlib.h, string.h, pthread.h, dirent.h. В программе используются следующие системные и библиотечные вызовы:

1. **opendir, readdir, closedir** – для открытия, обхода, и закрытия каталога /proc/<PID>/task, содержащего файлы с информацией о потоках процесса программы.
2. **getpid** – для получения PID процесса.
3. **read** – для чтения матрицы из файла.
4. **write** – для записи матрицы в файл после выполнения медианного фильтра.
5. **open** –для открытия файла с bmp-изображением (содержит матрицу целых чисел).
6. **close –** для закрытия файла с матрицей.
7. **pthread\_create –** для создания нового потока выполнения программы (в качестве аргумента передается функция которая будет запущена в новом потоке и её аргументы).
8. **pthread\_join –** для блокирующего ожидания конца выполнения потока по переданному ей идентификатору.
9. **pthread\_exit –** для выхода (завершения работы) потока.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы основных вызовов библиотеки pthread.
2. Написать функцию проверки корректности bmp изображения: bmp изображение считается корректным, если на пиксель приходится 8 бит, и не используется сжатие (256 цветный режим).
3. Написать функцию медианного фильтра.
4. Написать логику главного потока программы, принимающего аргументы командной строки, и запускающего функцию медианного фильтра в новом потоке с аргументами, ограничивающими его применение только для некоторых строк матрицы: для случая N потоков, i-ый поток проводит операцию медианного фильтра для i-ой строки, (n + i)-ой строки, (2n + i)-ой строки и.т.д. Синхронизация потов и отсутствие гонок данных достигается благодаря тому что все операции изменения производятся не в матрице, а записываются во временную матрицу.
5. Отладить программу и протестировать на основных тестах.

**Основные файлы программы**

**main.c**

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/time.h>

#include <sys/syscall.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <dirent.h>

#define NUM\_OF\_THREADS 128

#define MAX\_WIN\_W 8

#define MAX\_WIN\_H 8

struct bmpFileHeader {

    u\_int16\_t magic;

    u\_int32\_t size;

    u\_int32\_t reserved;

    u\_int32\_t offset;

} \_\_attribute\_\_((packed));

struct bmpInfo {

    u\_int32\_t size;

    u\_int32\_t width;

    u\_int32\_t height;

    u\_int16\_t planes;

    u\_int16\_t bitcount;

    u\_int32\_t compression;

    u\_int32\_t dataSize;

} \_\_attribute\_\_((packed));

struct pdata {

    u\_int8\_t v;

} \_\_attribute\_\_((packed));

struct args {

    struct pdata\* in;

    struct pdata\* out;

    u\_int32\_t line;

    u\_int32\_t increment;

    u\_int32\_t width;

    u\_int32\_t height;

};

void countingSort(int\* *array*, int *n*) {

    int c[256] = { 0 };

    int k = 256;

    for (int i = 0; i < *n*; i++)

        c[*array*[i]]++;

    int b = 0;

    for (int i = 0; i < k; i++)

        for (int j = 0; j < c[i]; j++)

*array*[b++] = i;

}

int compare(const void\* *arg1*, const void\* *arg2*) {

    return \*((int\*)(*arg1*)) < \*((int\*)(*arg2*));

}

struct pdata median(struct pdata *in*[MAX\_WIN\_H][MAX\_WIN\_W]) {

    int a[MAX\_WIN\_H \* MAX\_WIN\_W];

    for (int i = 0; i < MAX\_WIN\_H; ++i)

        for (int j = 0; j < MAX\_WIN\_W; ++j)

            a[i \* MAX\_WIN\_W + j] = *in*[i][j].v;

    countingSort(a, MAX\_WIN\_H \* MAX\_WIN\_W);

    struct pdata x;

    x.v = a[MAX\_WIN\_W \* MAX\_WIN\_H / 2];

    return x;

}

void\* medianFilter(void\* *arg*) {

    struct args\* a = (struct args\*)*arg*;

    int ex = MAX\_WIN\_W / 2;

    int ey = MAX\_WIN\_H / 2;

    struct pdata colors[MAX\_WIN\_H][MAX\_WIN\_W];

    for (int y = a->line; y < a->height; y += a->increment) {

        for (int x = 0; x < a->width; ++x) {

            for (int fy = 0; fy < MAX\_WIN\_H; ++fy) {

                for (int fx = 0; fx < MAX\_WIN\_W; ++fx) {

                    if ((y + fy - ey >= 0) && (x + fx - ex >= 0)) {

                        colors[fy][fx] = a->in[(y + fy - ey) \* a->width + (x + fx - ex)];

                    }

                }

            }

            a->out[y \* a->width + x] = median(colors);

        }

    }

    pthread\_exit(NULL);

}

void countThreads() {

    DIR\* dir;

    struct dirent\* entry;

    int pid = getpid();

    char dirname[256];

    sprintf(dirname, "/proc/%d/task", pid);

    int c = -3;

    if ((dir = opendir(dirname)) == NULL)

        perror("opendir() error");

    else {

        while ((entry = readdir(dir)) != NULL) c++;

        closedir(dir);

    }

    char buffer[256];

    sprintf(buffer, "Number of Threads: %d\n", c);

    write(0, buffer, strlen(buffer));

}

int t = 0;

void filterImage(struct pdata\* *in*, struct pdata\* *out*, u\_int32\_t *w*, u\_int32\_t *h*, u\_int32\_t *numOfThreads*) {

    struct args arg[NUM\_OF\_THREADS];

    pthread\_t handles[NUM\_OF\_THREADS];

    for (int i = 0; i < *numOfThreads*; ++i) {

        arg[i].in = *in*;

        arg[i].out = *out*;

        arg[i].line = i;

        arg[i].increment = *numOfThreads*;

        arg[i].width = *w*;

        arg[i].height = *h*;

        if (pthread\_create(&handles[i], NULL, medianFilter, &arg[i]) != 0)

            exit(-20);

    }

    if (!t) {

        countThreads();

        t = 1;

    }

    for (int i = 0; i < *numOfThreads*; ++i) {

        int res = pthread\_join(handles[i], NULL);

        if (res != 0)

            exit(-21);

    }

}

int parseBMPheader(int *fd*, struct bmpFileHeader\* *hdr*, struct bmpInfo\* *bmp*) {

    u\_int32\_t ret;

    if ((ret = read(*fd*, *hdr*, sizeof(\**hdr*))) != sizeof(\**hdr*))

        return -1;

    if (*hdr*->magic != 0x4D42)

        return -2;

    if (*hdr*->reserved != 0x00000000)

        return -2;

    if ((ret = read(*fd*, *bmp*, sizeof(\**bmp*))) != sizeof(\**bmp*))

        return -3;

    if (*bmp*->bitcount != 8) {

        perror("Unsupported bitsize!");

        return -1;

    }

    return 0;

}

long long getTime() {

    struct timeval te;

    gettimeofday(&te, NULL);

    long long milliseconds = te.tv\_sec \* 1000LL + te.tv\_usec / 1000;

    return milliseconds;

}

int main(int *argc*, char\* *argv*[]) {

    setbuf(stdout, NULL);

    u\_int32\_t ret;

    if (*argc* < 4)

        exit(1);

    int fd = open(*argv*[1], O\_RDWR);

    if (fd < 0)

        exit(fd);

    struct bmpFileHeader hdr;

    struct bmpInfo bmp;

    if (ret = parseBMPheader(fd, &hdr, &bmp))

        exit(ret);

    u\_int32\_t k = strtol(*argv*[2], NULL, 10);

    if (k > 100) {

        perror("Unsupported \'K\' value!");

        exit(-11);

    }

    u\_int32\_t thr = strtol(*argv*[3], NULL, 10);

    if (thr > NUM\_OF\_THREADS) {

        perror("Unsupported number of threads!");

        exit(-11);

    }

    if (lseek(fd, hdr.offset, SEEK\_SET) != hdr.offset)

        exit(-3);

    struct pdata\* inputData = malloc(bmp.dataSize);

    struct pdata\* outputData = malloc(bmp.dataSize);

    struct pdata\* tmp;

    if ((ret = read(fd, inputData, bmp.dataSize)) != bmp.dataSize) {

        perror("Bad read!");

        exit(-4);

    }

    double start, end;

    double dif;

    start = getTime();

    for (int i = 0; i < k; ++i) {

        filterImage(inputData, outputData, bmp.width, bmp.height, thr);

        tmp = inputData;

        inputData = outputData;

        outputData = tmp;

    }

    end = getTime();

    dif = end - start;

    char buffer[256];

    sprintf(buffer, "Time: %lf seconds.\n", dif / 1000.0);

    write(0, buffer, strlen(buffer));

    if (lseek(fd, hdr.offset, SEEK\_SET) != hdr.offset) {

        exit(-5);

    }

    if ((ret = write(fd, outputData, bmp.dataSize)) != bmp.dataSize) {

        perror("Bad write!");

        exit(-6);

    }

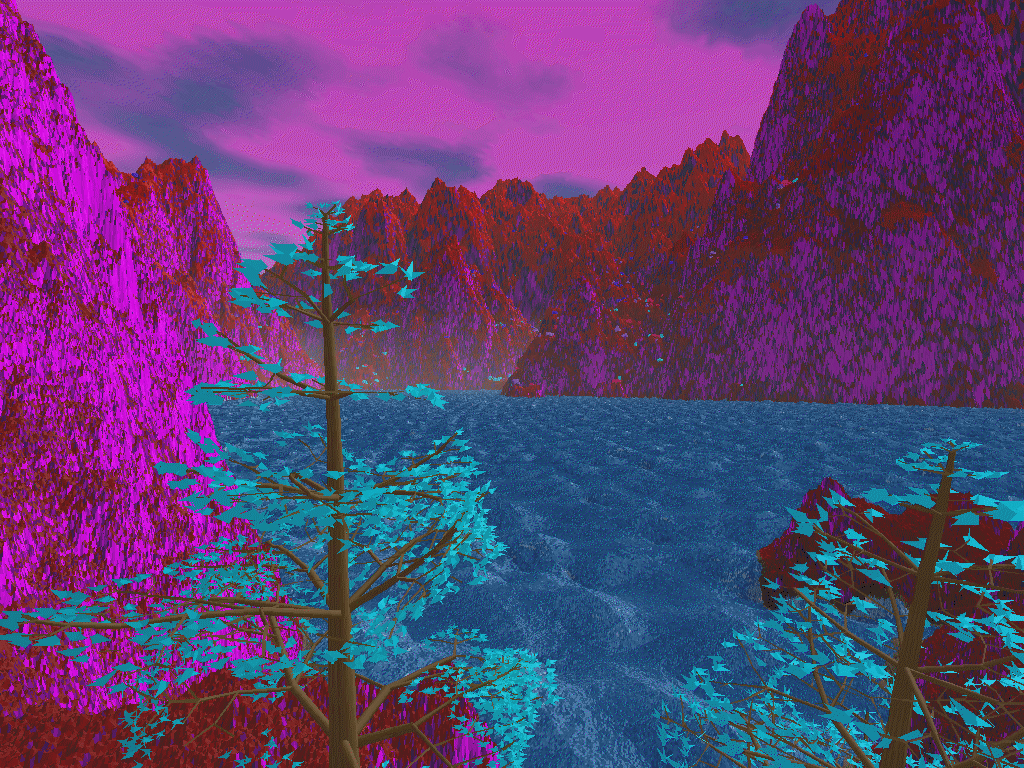
    close(fd);

    exit(0);

}

**Пример работы**

$ display ./LAND.BMP



$ make

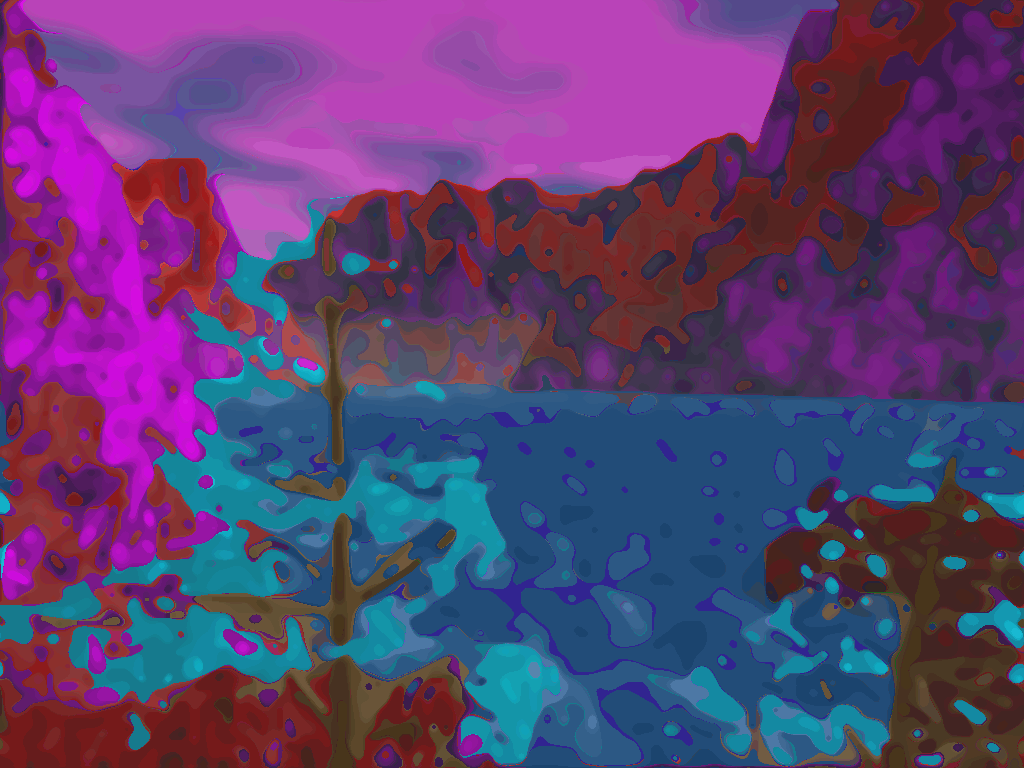
gcc -Werror -pedantic main.c -lpthread -o main

$ ./main ./LAND.BMP 8 4

Number of Threads: 4

Time: 1.956000 seconds.

$ display ./LAND.BMP



$ ./main ./LAND.BMP 8 1

Number of Threads: 1

Time: 5.909000 seconds.

**Репозиторий**

https://github.com/YurokSov/os\_lab\_3

**Вывод**

Выполнив эту лабораторную работу, я приобрел навыки программирования простых программ, использующих несколько потоков для ускорения вычислений. В моей программе, при использовании четырех потоков, удалось добиться скорости выполнения, превышающей скорость выполнения алгоритма на одном потоке примерно в три раза. Также я познакомился с простейшим форматом хранения изображения – bmp.