Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №5 по курсу**

**«Операционные системы»**

**ДИНАМИЧЕСКИЕ БИБЛИОТЕКИ**

Студент: Ильин И. О

Группа: М8О–206Б–20

Вариант: 11

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Создание динамических библиотек
* Создание программ, которые используют функции динамических библиотек

## 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 программ

ой 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 фильтры эрозии и наращивания на матрицу, состоящую из вещественных чисел. На выходе получается 2 результирующие матрицы

**Общие сведения о программе**

Программа компилируется из файла lab3.c. Также используется заголовочные файлы: stdio.h, unistd.h, stdlib.h, ctype.h, stdbool.h, pthread.h, time.h, dirent.h, sys/time.h. В программе используются следующие системные вызовы:

1. **gettimeofday** ­­–– позволяет получить текущее время. Принимает в качестве аргументов структуру timeval, в которую записывает результат работы, и переменную для коррекции времени. В случае успеха возвращает 0.
2. **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.
3. **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.
4. **pthread\_exit ––** завершает вызываемый поток. В качестве аргумента принимает значение, которое вернется при завершении потока. Функция всегда завершается успехом.
5. **read ––** предназначена для чтения какого-то числа байт из файла, принимает в качестве аргументов файловый дескриптор, буфер, в который будут записаны данные и число байт. В случае успеха вернет число прочитанных байт, иначе -1.
6. **write ––** предназначена для записи какого-то числа байт в файл, принимает в качестве аргументов файловый дескриптор, буфер, из которого будут считаны данные для записи и число байт. В случае успеха вернет число записанных байт, иначе -1.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы с потоками в C.
2. Написать вспомогательные функции для работы с матрицами.
3. С помощью принципа разбиения задачи на маленькие подзадачи, реализовать функцию, которая будет проводить частичную обработку данных, для ее запуска в мальтипоточном режиме.
4. Реализовать функцию, которая будет создавать потоки, запускать потоковую функцию, управлять потоками.
5. Реализовать обработку системных ошибок согласно заданию

**Основные файлы программы**

**lab3.c:**

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

#include <stdbool.h>

#include <unistd.h>

#include <pthread.h>

#include <time.h>

#include <dirent.h>

#include <sys/time.h>

// 11. Наложить K раз фильтры эрозии и наращивания на матрицу, состоящую из вещественных чисел. На выходе получается 2 результирующие матрицы

// Ввод - размеры матрицы, сама матрица

// Окно фильтров 3х3

void print\_usage(char\* cmd) {

printf("Usage: %s [-threads num]\n", cmd);

}

bool read\_matrix(float\* matrix, size\_t rows, size\_t cols) {

for (size\_t i = 0; i < rows; ++i) {

for (size\_t j = 0; j < cols; ++j) {

if (scanf("%f", &matrix[i\*cols + j]) != 1) {

perror("Error while reading matrix");

return false;

}

//scanf("%f", &matrix[i\*cols + j]);

//printf("=== matrix[%ld] = %f\n", i\*cols + j, matrix[i\*cols + j]);

}

}

return true;;

}

bool print\_matrix(float\* matrix, size\_t rows, size\_t cols) {

for (size\_t i = 0; i < rows; ++i) {

for (size\_t j = 0; j < cols; ++j) {

printf("%.20g ", matrix[i\*cols + j]);

}

printf("\n");

}

return false;

}

void copy\_matrix(float\* from, float\* to, size\_t rows, size\_t cols) {

for (size\_t i = 0; i < rows; ++i) {

for (size\_t j = 0; j < cols; ++j) {

to[i\*cols + j] = from[i\*cols + j];

}

}

}

typedef enum {

EROISON,

DILATION

} filter\_type;

typedef struct {

int thread\_num;

int th\_count;

int rows;

int cols;

int w\_dim;

float\* matrix1;

float\* result1;

float\* matrix2;

float\* result2;

} thread\_arg;

// Функция, которая будет выполняться в потоках

// Происзводит попоточную обработку строк матрицы

void\* edit\_line(void\* argument) {

thread\_arg\* args = (thread\_arg\*)argument;

const int thread\_num = args->thread\_num;

const int th\_count = args->th\_count;

const int rows = args->rows;

const int cols = args->cols;

int offset = args->w\_dim / 2;

const float\* matrix1 = args->matrix1;

const float\* matrix2 = args->matrix2;

float\* result1 = args->result1;

float\* result2 = args->result2;

// Либо 2 одновременно либо 2 ф-ии

//printf("\n=== IN THREAD %d ===\n", thread\_num);

// printf("offset = %d\n", offset);

for (int th\_row = thread\_num; th\_row < rows; th\_row += th\_count) {

//printf("THREAD %d ROW %d\n", thread\_num, th\_row);

for (int th\_col = 0; th\_col < cols; ++th\_col) {

//printf(" th\_col = %d\n", th\_col);

float max = matrix1[th\_row\*cols + th\_col];

float min = matrix2[th\_row\*cols + th\_col];

for (int i = th\_row - offset; i < th\_row + offset + 1; ++i) {

for (int j = th\_col - offset; j < th\_col + offset + 1; ++j) {

float curr1, curr2;

if ((i < 0) || (i >= rows) || (j < 0) || (j >= cols)) {

curr1 = 0;

curr2 = 0;

} else {

curr1 = matrix1[i\*cols + j];

curr2 = matrix2[i\*cols + j];

}

//printf("[%d][%d] = %f ", i, j, curr1);

if (curr1 > max) {

max = curr1;

}

if (curr2 < min) {

min = curr2;

}

}

//printf("\n");

}

result1[th\_row\*cols + th\_col] = max;

result2[th\_row\*cols + th\_col] = min;

}

//printf("\n");

}

pthread\_exit(NULL); // Заканчиваем поток

}

void put\_filters(float\* matrix, size\_t rows, size\_t cols, size\_t w\_dim, float\* res1, float\* res2, int filter\_cnt, int th\_count) {

float\* tmp1 = (float\*)malloc(rows \* cols \* sizeof(float));

if (!tmp1) {

perror("Error while allocating matrix\n");

exit(1);

}

float\* tmp2 = (float\*)malloc(rows \* cols \* sizeof(float));

if (!tmp2) {

perror("Error while allocating matrix\n");

exit(1);

}

copy\_matrix(matrix, tmp1, rows, cols);

copy\_matrix(matrix, tmp2, rows, cols);

pthread\_t ids[th\_count];

thread\_arg args[th\_count];

for (int k = 0; k < filter\_cnt; ++k) {

for (int i = 0; i < th\_count; ++i) {

args[i].thread\_num = i;

args[i].th\_count = th\_count;

args[i].rows = rows;

args[i].cols = cols;

args[i].w\_dim = w\_dim;

args[i].matrix1 = tmp1;

args[i].result1 = res1;

args[i].matrix2 = tmp2;

args[i].result2 = res2;

if (pthread\_create(&ids[i], NULL, edit\_line, &args[i]) != 0) {

perror("Can't create a thread.\n");

}

}

for(int i = 0; i < th\_count; ++i) {

if (pthread\_join(ids[i], NULL) != 0) {

perror("Can't wait for thread\n");

}

}

if (filter\_cnt != 1) {

float\* swap = res1;

res1 = tmp1;

tmp1 = swap;

swap = res2;

res2 = tmp2;

tmp2 = swap;

// copy\_matrix(res1, tmp1, rows, cols);

// copy\_matrix(res2, tmp2, rows, cols);

}

}

free(tmp1);

free(tmp2);

}

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

int pid = getpid();

printf("PID <%d>\n", pid);

int threads = 1;

if (argc == 3) {

threads = atoi(argv[2]);

} else if (argc != 1) {

print\_usage(argv[0]);

return 0;

}

printf("Total thread = %d\n", threads);

int rows;

int cols;

printf("Enter matrix dimensions:\n");

scanf("%d", &cols);

scanf("%d", &rows);

float\* matrix = (float\*)malloc(rows \* cols \* sizeof(float));

float\* res1 = (float\*)malloc(rows \* cols \* sizeof(float));

float\* res2 = (float\*)malloc(rows \* cols \* sizeof(float));

if (!matrix || !res1 || !res2) {

perror("Error while allocating matrix\n");

return 1;

}

read\_matrix(matrix, rows, cols);

int w\_dim;

printf("Enter window dimension:\n");

scanf("%d", &w\_dim);

if (w\_dim % 2 == 0) {

perror("Window dimension must be an odd number\n");

return 1;

}

printf("Enter K:\n");

int k;

scanf("%d", &k);

struct timeval start, end;

gettimeofday(&start, NULL);

put\_filters(matrix, rows, cols, w\_dim, res1, res2, k, threads);

gettimeofday(&end, NULL);

long sec = end.tv\_sec - start.tv\_sec;

long microsec = end.tv\_usec - start.tv\_usec;

if (microsec < 0) {

--sec;

microsec += 1000000;

}

long elapsed = sec\*1000000 + microsec;

printf("DILATION:\n");

print\_matrix(res1, rows, cols);

printf("EROISON:\n");

print\_matrix(res2, rows, cols);

printf("TOTAL TIME: %ld ms\n", elapsed);

free(res1);

free(res2);

free(matrix);

return 0;

}

**Пример работы**

[Запуск тестов в терминале, работа с программой вручную]

**Вывод**

[Очень важный раздел. Вода здесь может негативно повлиять на оценку]