Министерство образования и науки Российской Федерации

Московский авиационный институт  
(национальный исследовательский университет)

Кафедра №304

Отчет по лабораторной работе №5

«**Потоки исполнения процессов в Linux.**

**Мультипоточный фильтр Собеля**»

по дисциплине «Операционные системы»

Выполнил

Студент группы 3О-310Б

Сомов Д.Н.

Приняли

ассистент каф.304

Балян А.В.

ассистент каф.304

Аляутдин Р.Т.

Москва, 2017

**Задание:**

Ознакомиться с функциями Linux из библиотеки pthread.h, позволяющими создавать потоки исполнения внутри одного процесса, запускать в них функции; останавливать и соединять потоки. В качестве практического применения потоков написать многопоточную программу, реализующую оператор Собеля для фильтрации изображений (выделение контуров на изображении). Использовать таймер для определения времени, затраченного на применение оператора Собеля к изображению (для наглядности пользы многопоточности). Проверить написанную программу с помощью Memory Sanitizer, checkpatch.pl. Использовать систему контроля версий git и удаленный репозиторий на GitHub.

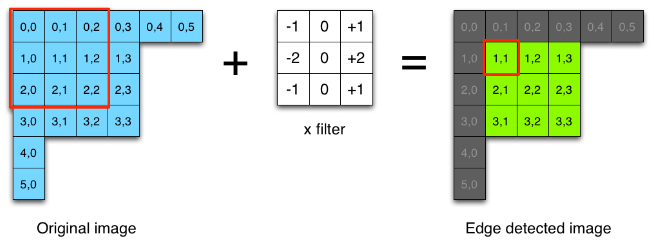
**Описание:**

Оператор Собеля — дискретный дифференциальный оператор, вычисляющий приближённое значение градиента яркости изображения. Результатом применения оператора Собеля в каждой точке изображения является вектор градиента яркости в этой точке. Используется в области обработки изображений, в частности, часто применяется в алгоритмах выделения границ. Если изображение обработать фильтром Собеля, на выходном изображении будут выделены границы объектов.

Преобразование осуществляется с помощью двух ядер свертки (переменные kX и kY в программе ниже); при перемножении матрицы значений пикселей 3х3 черно-белого исходного изображения и каждого ядра свертки получаются два значения Gx и Gy, которые затем используются в формуле:

pixsobel и есть значение соответствующего пикселя выходного изображения.

Пример для х-ядра:



Для реализации многопоточности (с целью ускорения вычислений вышеуказанных преобразований) была введена функция, которую в первую очередь исполняет каждый созданный поток; она определяет высоту и начало «полосы» изображения (все полосы одинаковые по объему), которую должен будет преобразовать этот конкретный поток, основываясь на заданном общем количестве потоков. После этого запускается таймер, и основной поток начинает ждать завершения всех потоков, созданных для фильтрации. Как только все потоки завершатся, таймер останавливается, и фильтрованное изображение сохраняется на диск.

**Исходный код (на GitHub: /NotSoOld/Lab5-sobel ):**

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* Programming language – C

\* Author – Dmitriy Somov, 3O-310B

\*

\* **File: sobel.c**

\*/

#define \_REENTRANT

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <pthread.h>

#include <time.h>

#define STB\_IMAGE\_IMPLEMENTATION

#define STBI\_NO\_HDR

#define STBI\_NO\_LINEAR

#include "stb/stb\_image.h"

#define STB\_IMAGE\_WRITE\_IMPLEMENTATION

#include "stb/stb\_image\_write.h"

#define byte unsigned char

#define BUFSIZE 512

// Uncomment the following to see multithreading debug information:

//#define DEBUG\_THREADING

int kX[3][3] = {{-1, 0, 1},

{-2, 0, 2},

{-1, 0, 1}};

int kY[3][3] = {{ 1, 2, 1},

{ 0, 0, 0},

{-1,-2,-1}};

byte \*img; // Input image data pointer

byte \*sobelImg; // Output image data pointer

int threadsNum, x, y;

// Converts RGB pixel to grayscale pixel.

byte ToGrayscale(byte r, byte g, byte b)

{

return (byte)((r + g + b) / 3);

}

// Calculates one component for Sobel operator

// from given matrix and image data.

int Convolute(int kernel[3][3], byte \*imgData, int x)

{

return

kernel[0][0] \* ToGrayscale(imgData[-x-3], imgData[-x-2], imgData[-x-1]) +

kernel[0][1] \* ToGrayscale(imgData[-3], imgData[-2], imgData[-1]) +

kernel[0][2] \* ToGrayscale(imgData[x-3], imgData[x-2], imgData[x-1]) +

kernel[1][0] \* ToGrayscale(imgData[-x], imgData[-x+1], imgData[-x+2]) +

kernel[1][1] \* ToGrayscale(imgData[0], imgData[1], imgData[2]) +

kernel[1][2] \* ToGrayscale(imgData[x], imgData[x+1], imgData[x+2]) +

kernel[2][0] \* ToGrayscale(imgData[-x+3], imgData[-x+4], imgData[-x+5]) +

kernel[2][1] \* ToGrayscale(imgData[3], imgData[4], imgData[5]) +

kernel[2][2] \* ToGrayscale(imgData[x+3], imgData[x+4], imgData[x+5]);

}

void ShowManual(void)

{

printf("Usage: sobel file\_name number\_of\_threads\n");

printf("ERROR: not enough arguments. Stopping.\n");

exit(1);

}

// Start method for each thread.

void \*SobelThread(void \*id)

{

int i, j, Gx, Gy;

int strip = (int)ceil(y / threadsNum);

int indent = strip \* (intptr\_t)id;

int stop = (strip+indent == y) ? y-1 : strip+indent;

indent += ((intptr\_t)id == 0) ? 1 : 0;

#ifdef DEBUG\_THREADING

printf("id = %li, indent = %i, stop = %i\n", (intptr\_t)id, indent, stop);

#endif

// We're working on image only inside given strip.

// Size of strip is calculated according to the number of threads.

for (i = indent; i < stop; i++) {

for (j = 3; j < x\*3-3; j += 3) {

Gx = Convolute(kX, img + i\*x\*3+j, x\*3);

Gy = Convolute(kY, img + i\*x\*3+j, x\*3);

sobelImg[(i-1)\*(x-2)+ j/3 - 1)] = (byte)(sqrt(Gx\*Gx + Gy\*Gy));

}

}

pthread\_exit(NULL);

}

// Creates the name of the output image.

void InsertSuffix(char \*inputName, char \*outputName)

{

int i, j;

for (i = strlen(inputName)-1; i >= 0; i--) {

if (inputName[i] == '.')

break;

}

for (j = 0; j < i; j++)

outputName[j] = inputName[j];

strcat(outputName, "\_sobel.png");

}

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

{

int n, k;

char outName[BUFSIZE];

double tbefore, tafter;

struct timespec before;

struct timespec after;

if (argc != 3)

ShowManual();

// Load image.

img = stbi\_load(argv[1], &x, &y, &n, 0);

if (img == NULL) {

printf("Failed to load the image %s! Stopping.\n", argv[1]);

exit(2);

}

printf("Input image: %ix%i, %i pixel contents\n", x, y, n);

memset(outName, '\0', BUFSIZE);

InsertSuffix(argv[1], outName);

printf("Calculating Sobel operations...\n");

sobelImg = (byte \*)calloc((x-2)\*(y-2), 1);

// Here we start multithreading.

threadsNum = atoi(argv[2]);

if (threadsNum == 0) {

printf("Cannot parse number of threads '%s'! Stopping.\n", argv[2]);

exit(3);

}

pthread\_t threads[threadsNum];

printf("Timer started.\n");

// Get time before operations.

if (clock\_gettime(CLOCK\_REALTIME, &before) == -1) {

perror("Failed to get time before threading started");

exit(7);

}

for (k = 0; k < threadsNum; k++) {

// Create another thread and start computations in it.

n = pthread\_create(&(threads[k]),

NULL,

SobelThread,

(void \*)(intptr\_t)k);

if (n != 0) {

printf("Failed to create thread #%i. Stopping.\n", k);

perror("Purpose");

exit(4);

}

}

k = threadsNum - 1;

for (; k >= 0; k--) {

// After thread had finished its job, it'll be joined.

n = pthread\_join(threads[k], NULL);

if (n != 0) {

printf("Failed to join thread #%i. Stopping.\n", k);

perror("Purpose");

exit(5);

}

#ifdef DEBUG\_THREADING

else {

printf("Joined thread #%i\n", k);

}

#endif

}

// Get time after operations.

if (clock\_gettime(CLOCK\_REALTIME, &after) == -1) {

perror("Failed to get time after threading finished");

exit(8);

}

printf("Timer stopped.\n");

// Calculate and print elapsed time.

tbefore = before.tv\_sec + before.tv\_nsec \* 0.000000001;

tafter = after.tv\_sec + after.tv\_nsec \* 0.000000001;

tafter -= tbefore;

printf("Elapsed time: %.3lf s\n", tafter);

// Write filtered image to disk.

printf("Saving image after Sobel operator as %s...\n", outName);

n = stbi\_write\_png(outName, x-2, y-2, 1, sobelImg, 0);

if (n == 0) {

printf("Failed to save image to disk!\n");

exit(6);

}

stbi\_image\_free(img);

free(sobelImg);

printf("Done.\n");

exit(0);

}

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

# Author – Dmitriy Somov, 3O-310B

#

# **File: makefile**

regular:

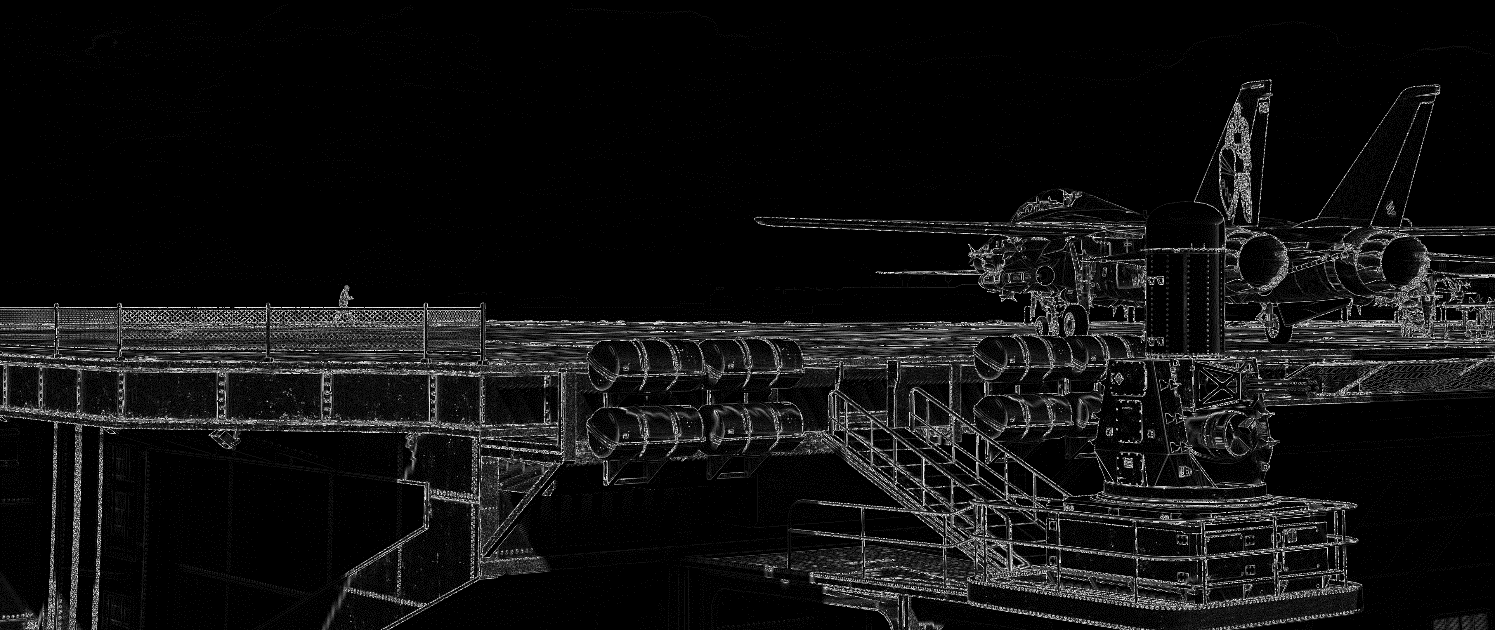
gcc -D\_REENTERANT -o sobel sobel.c -lm -lpthread

memsan:

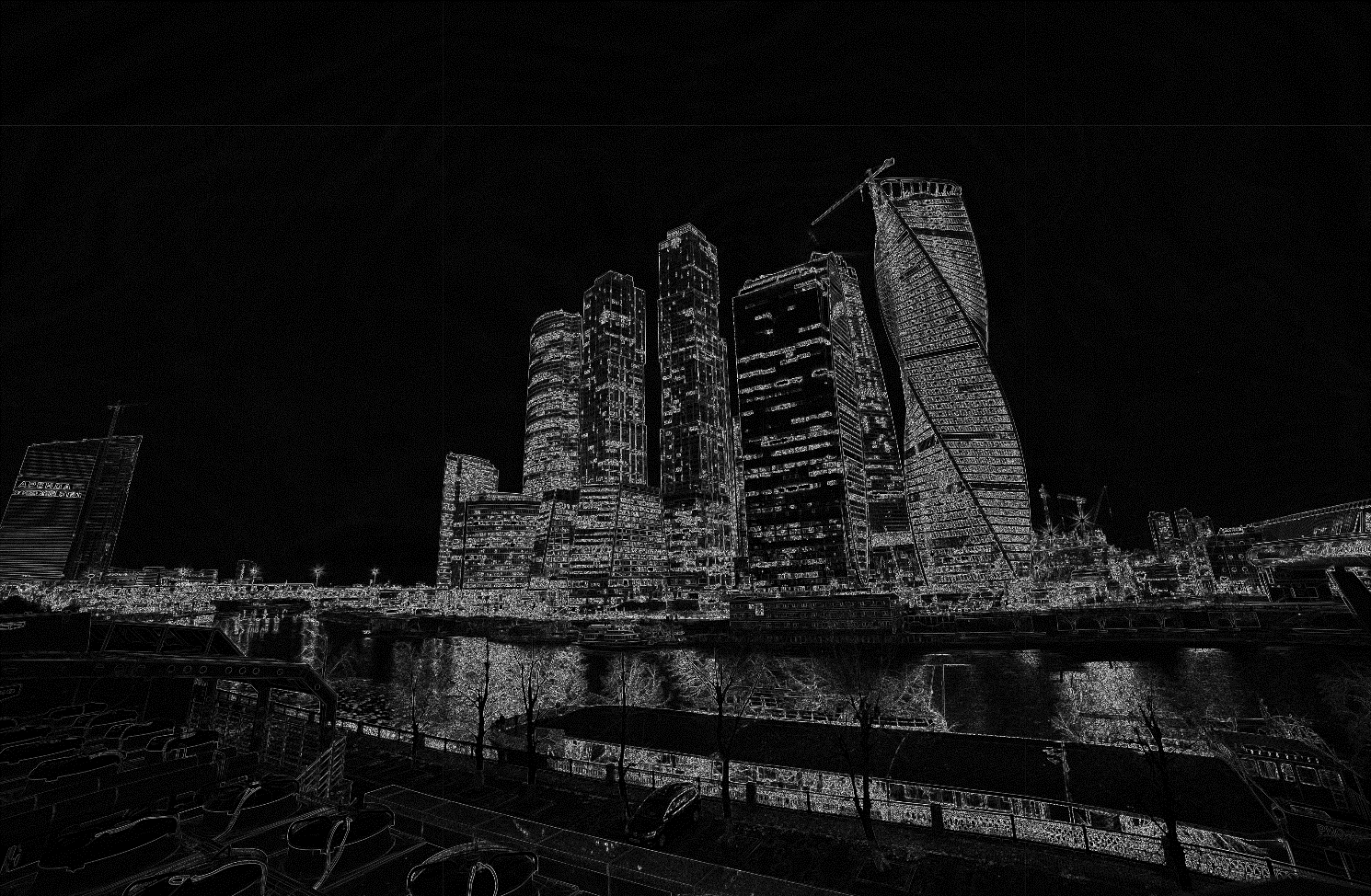
gcc -D\_REENTERANT -o sobel\_m sobel.c -fsanitize=address -lm -lpthread

**Тестируемые изображения до и после фильтрации:**

****

****

****

****

****

****

**Результаты замеров времени исполнения операций фильтрации:**

Input image: 6000x4000, 3 pixel contents

./sobel images/3.jpg 1

Elapsed time: 3.500 s

./sobel images/3.jpg 2

Elapsed time: 1.818 s

./sobel images/3.jpg 3

Elapsed time: 1.324 s

./sobel images/3.jpg 4

Elapsed time: 1.089 s

./sobel images/3.jpg 5

Elapsed time: 0.896 s

./sobel images/3.jpg 6

Elapsed time: 0.761 s

./sobel images/3.jpg 7

Elapsed time: 0.821 s

./sobel images/3.jpg 8

Elapsed time: 0.896 s

./sobel images/3.jpg 9

Elapsed time: 0.804 s

./sobel images/3.jpg 10

Elapsed time: 0.817 s

./sobel images/3.jpg 15

Elapsed time: 0.757 s

./sobel images/3.jpg 20

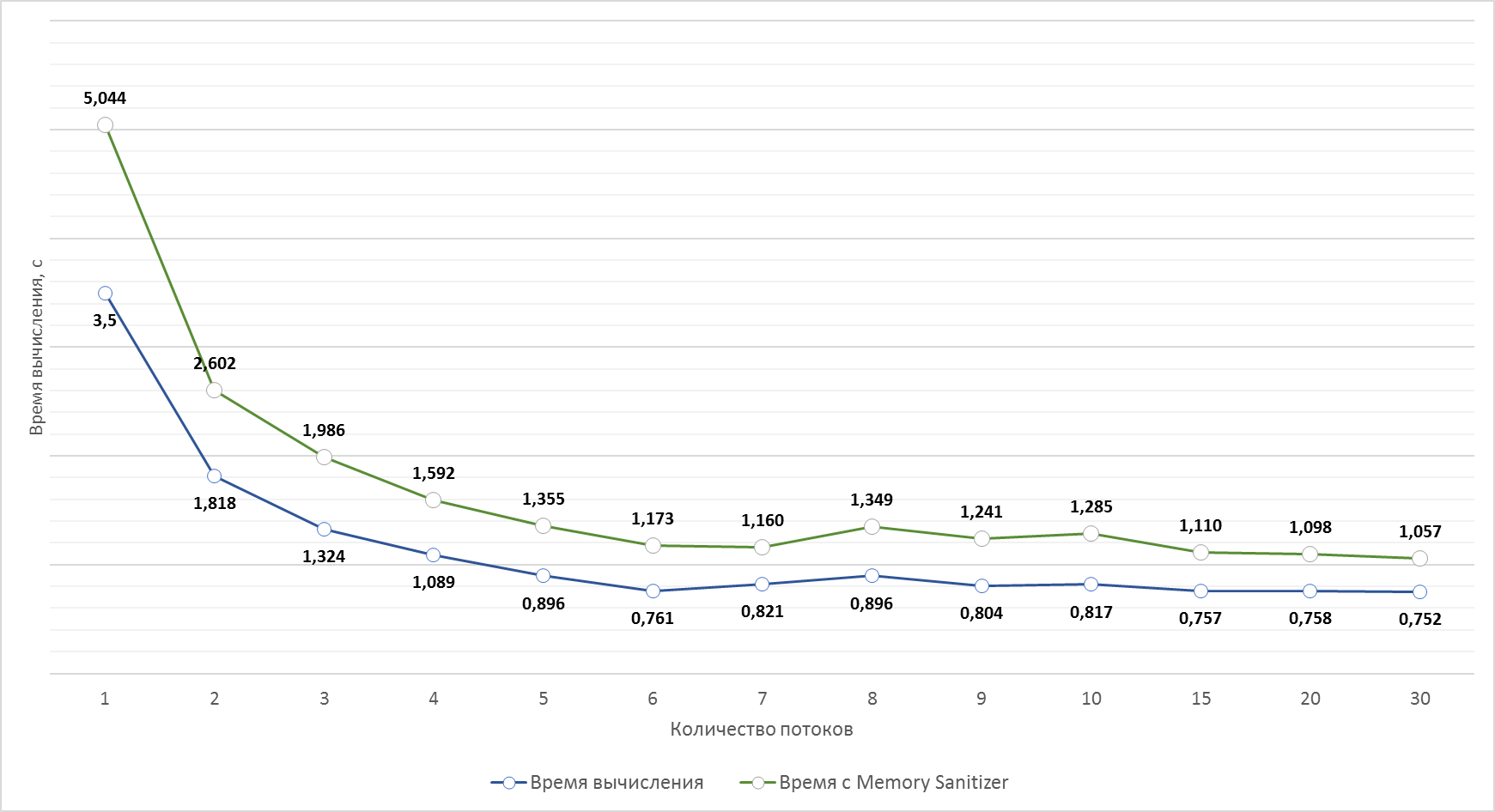
Elapsed time: 0.758 s

./sobel images/3.jpg 30

Elapsed time: 0.752 s

**График зависимости времени исполнения от количества потоков:**

(количество логических ядер процессора машины - 6)

****