

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

**Лабораторная работа №3 по курсу  
«Операционные системы»**

Студент: Боев Савелий Сергеевич  
Группа: М8О-207Б-21  
Вариант: 11  
Преподаватель: Миронов Евгений Сергеевич  
Оценка: \_\_\_\_\_  
Дата: \_\_\_\_\_  
Подпись: \_\_\_\_\_

Москва, 2022

## Содержание

Репозиторий.....	3
Постановка задачи .....	3
Цель работы .....	3
Задание .....	3
Исходный код.....	4
Демонстрация работы программы .....	9
Замеры времени.....	9
Выводы.....	12

# Репозиторий

<https://github.com/IamNoobLEL/Labs-OSi>

## Постановка задачи

### Цель работы

Изучение операционных систем

### Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке 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 программой 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.

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

## Исходный код

```
1. #include <stdio.h>
2. #include <ctype.h>
3. #include <stdlib.h>
4. #include <stdbool.h>
5. #include <unistd.h>
6. #include <pthread.h>
7. #include <time.h>
8. #include <dirent.h>
9. #include <sys/time.h>
10.
11.
12. void print_usage(char* cmd) {
13.     printf("Usage: %s [-threads num]\n", cmd);
14. }
15.
16. bool read_matrix(float* matrix, size_t rows, size_t cols) {
17.     for (size_t i = 0; i < rows; i++) {
18.         for (size_t j = 0; j < cols; j++) {
19.             if (scanf("%f", &matrix[i * cols + j]) != 1) {
20.                 perror("Error while reading matrix");
21.                 return false;
22.             }
23.             //scanf("%f", &matrix[i*cols + j]);
24.             //printf("=== matrix[%ld] = %f\n", i*cols + j, matrix[i*cols + j]);
25.         }
26.     }
27.     return true;;
28. }
29.
30. bool print_matrix(float* matrix, size_t rows, size_t cols) {
31.     for (size_t i = 0; i < rows; i++) {
32.         for (size_t j = 0; j < cols; j++) {
33.             printf("%.20g ", matrix[i * cols + j]);
34.         }
35.         printf("\n");
36.     }
37.     return false;
38. }
39.
40. void copy_matrix(float* from, float* to, size_t rows, size_t cols) {
41.     for (size_t i = 0; i < rows; i++) {
42.         for (size_t j = 0; j < cols; j++) {
43.             to[i * cols + j] = from[i * cols + j];
44.         }
45.     }
46. }
47.
48. typedef struct {
```

```

49.     int thread_num;
50.     int th_count;
51.     int rows;
52.     int cols;
53.     int w_dim;
54.     float** matrix1;
55.     float** result1;
56.     float** matrix2;
57.     float** result2;
58. } thread_arg;
59.
60. void* edit_line(void* argument) {
61.     thread_arg* args = (thread_arg*)argument;
62.     const int thread_num = args->thread_num;
63.     const int th_count = args->th_count;
64.
65.     const int rows = args->rows;
66.     const int cols = args->cols;
67.     int offset = args->w_dim / 2;
68.
69.     float** matrix1_ptr = args->matrix1;
70.     float** matrix2_ptr = args->matrix2;
71.     float** result1_ptr = args->result1;
72.     float** result2_ptr = args->result2;
73.
74.     const float* matrix1 = *matrix1_ptr;
75.     const float* matrix2 = *matrix2_ptr;
76.     float* result1 = *result1_ptr;
77.     float* result2 = *result2_ptr;
78.
79.     //printf("\n=== IN THREAD %d ===\n", thread_num);
80.     // printf("offset = %d\n", offset);
81.
82.     for (int th_row = thread_num; th_row < rows; th_row += th_count) {
83.         //printf("THREAD %d ROW  %d\n", thread_num, th_row);
84.         for (int th_col = 0; th_col < cols; th_col++) {
85.             // printf(" th_col = %d\n", th_col);
86.             float max = matrix1[th_row * cols + th_col];
87.             float min = matrix2[th_row * cols + th_col];
88.             for (int i = th_row - offset; i < th_row + offset + 1; i++) {
89.                 for (int j = th_col - offset; j < th_col + offset + 1; j++) {
90.                     float curr1, curr2;
91.                     if ((i < 0) || (i >= rows) || (j < 0) || (j >= cols)) {
92.                         curr1 = 0;
93.                         curr2 = 0;
94.                     } else {
95.                         curr1 = matrix1[i * cols + j];
96.                         curr2 = matrix2[i * cols + j];
97.                     }
98.                     // printf("[%d][%d] ", i, j);
99.                     if (curr1 > max) {

```

```

100.             max = curr1;
101.         }
102.         if (curr2 < min) {
103.             min = curr2;
104.         }
105.     }
106.     // printf("\n");
107. }
108.
109.     result1[th_row * cols + th_col] = max;
110.     result2[th_row * cols + th_col] = min;
111. }
112. //printf("\n");
113. }
114. pthread_exit(NULL); // Заканчиваем поток
115. }
116.
117. void put_filters(float** matrix_ptr, size_t rows, size_t cols, size_t w_dim,
float** res1_ptr, float** res2_ptr, int filter_cnt, int th_count) {
118.     float* tmp1 = (float*)malloc(rows * cols * sizeof(float));
119.     if (!tmp1) {
120.         perror("Error while allocating matrix\n");
121.         exit(1);
122.     }
123.     float** matrix1_ptr = &tmp1;
124.     float* tmp2 = (float*)malloc(rows * cols * sizeof(float));
125.     if (!tmp2) {
126.         perror("Error while allocating matrix\n");
127.         exit(1);
128.     }
129.     float** matrix2_ptr = &tmp2;
130.     copy_matrix(*matrix_ptr, tmp1, rows, cols);
131.     copy_matrix(*matrix_ptr, tmp2, rows, cols);
132.
133.     pthread_t ids[th_count];
134.     thread_arg args[th_count];
135.
136.     for (int k = 0; k < filter_cnt; k++) {
137.         for (int i = 0; i < th_count; i++) {
138.             args[i].thread_num = i;
139.             args[i].th_count = th_count;
140.             args[i].rows = rows;
141.             args[i].cols = cols;
142.             args[i].w_dim = w_dim;
143.             args[i].matrix1 = matrix1_ptr;
144.             args[i].result1 = res1_ptr;
145.             args[i].matrix2 = matrix2_ptr;
146.             args[i].result2 = res2_ptr;
147.
148.             if (pthread_create(&ids[i], NULL, edit_line, &args[i]) != 0) {
149.                 perror("Can't create a thread.\n");

```

```

150.         }
151.     }
152.
153.     for(int i = 0; i < th_count; i++) {
154.         if (pthread_join(ids[i], NULL) != 0) {
155.             perror("Can't wait for thread\n");
156.         }
157.     }
158.
159.     if (filter_cnt > 1) {
160.         float** swap = res1_ptr;
161.         res1_ptr = matrix1_ptr;
162.         matrix1_ptr = swap;
163.
164.         swap = res2_ptr;
165.         res2_ptr = matrix2_ptr;
166.         matrix2_ptr = swap;
167.     }
168. }
169.
170. free(tmp1);
171. free(tmp2);
172. }
173.
174. int main(int argc, char* argv[]) {
175.     printf("Enter K = ");
176.     int threads;
177.     scanf("%d", &threads);
178.
179.     if (argc == 3) {
180.         threads = atoi(argv[2]);
181.     } else if (argc != 1) {
182.         print_usage(argv[0]);
183.         return 0;
184.     }
185.
186.     int rows;
187.     int cols;
188.     printf("Enter matrix dimensions:\n");
189.     scanf("%d", &cols);
190.     scanf("%d", &rows);
191.     float* matrix = (float*)malloc(rows * cols * sizeof(float));
192.     float* res1 = (float*)malloc(rows * cols * sizeof(float));
193.     float* res2 = (float*)malloc(rows * cols * sizeof(float));
194.     if (!matrix || !res1 || !res2) {
195.         perror("Error while allocating matrix\n");
196.         return 1;
197.     }
198.     read_matrix(matrix, rows, cols);
199.
200.     int w_dim;

```

```

201.     printf("Enter window dimension:\n");
202.     scanf("%d", &w_dim);
203.     if (w_dim % 2 == 0) {
204.         perror("Window dimension must be an odd number\n");
205.         return 1;
206.     }
207.
208.     printf("Result \n");
209.     int k;
210.     scanf("%d", &k);
211.
212.     struct timeval start, end;
213.     gettimeofday(&start, NULL);
214.
215.     put_filters(&matrix, rows, cols, w_dim, &res1, &res2, k, threads);
216.
217.     gettimeofday(&end, NULL);
218.
219.     long sec = end.tv_sec - start.tv_sec;
220.     long microsec = end.tv_usec - start.tv_usec;
221.     if (microsec < 0) {
222.         --sec;
223.         microsec += 1000000;
224.     }
225.     long elapsed = sec*1000000 + microsec;
226.
227.
228.     printf("Dilation:\n");
229.     print_matrix(res1, rows, cols);
230.     printf("Erosion:\n");
231.     print_matrix(res2, rows, cols);
232.     printf("Total time: %ld ms\n", elapsed);
233.
234.     free(res1);
235.     free(res2);
236.     free(matrix);
237.     return 0;
238. }

```



## Демонстрация работы программы

```
savely@SavelyUBU: ~/CTon/OSI/Labs3
savely@SavelyUBU:~/CTon/OSI/Labs3$ ./main
Enter K = 1
Enter matrix dimensions:
5 5
1 0 1 1 1
1 1 1 0 1
1 1 1 1 1
0 1 0 1 1
1 1 1 0 1
Enter window dimension:
3 1
Result
Dilatation:
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
Erosion:
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
Total time: 660 ms
savely@SavelyUBU:~/CTon/OSI/Labs3$
```

## Замеры времени

[illegible]

1 поток





[illegible]

4 поток

[illegible]

5 потоков

## **Выводы**

Мною было освоено написании программы, создающей и производящей вычисления в нескольких потоках, а также синхронизация данных между этими потоками. Это позволяет серьезно сократить время выполнения некоторых задач, ибо действия выполняются не последовательно, а параллельно.