

Московский Авиационный Институт  
(Национальный Исследовательский Университет)  
Институт №8 “Компьютерные науки и прикладная математика”  
Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №2 по курсу**  
**«Операционные системы»**

Группа: М8О-211Б-23

Студент:

Тульчинский Г.С.  
Преподаватель: Бахарев В.Д.

Оценка:

---

Дата: 17.12.24

# 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 обработке использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение максимального количества потоков, работающих в один момент времени, 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 входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

**Вариант 13. Наложить K раз фильтр, использующий матрицу свертки, на матрицу, состоящую из вещественных чисел. Размер окна задается пользователем.**

## Общий метод и алгоритм решения

Использованные системные вызовы:

- `int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void*(*start_routine) (void*), void *arg)` - создаёт новый поток
- `int pthread_join(pthread_t threads, void ** value)` - Дождется завершения потока
- `int pthread_mutex_init(pthread_mutex_t *mutex)` - инициализирует мьютекс
- `int pthread_mutex_lock(pthread_mutex_t *mutex)` - захватывает мьютекс
- `int pthread_mutex_unlock(pthread_mutex_t *mutex)` - освобождает мьютекс
- `int pthread_mutex_destroy(pthread_mutex_t *mutex)` - уничтожает мьютекс

Программа получает на вход 4 аргумента – размер матрицы, размер фильтра, количество итераций и максимальное количество потоков. `filter` отвечает за то сколько раз программа будет фильтровать матрицу.

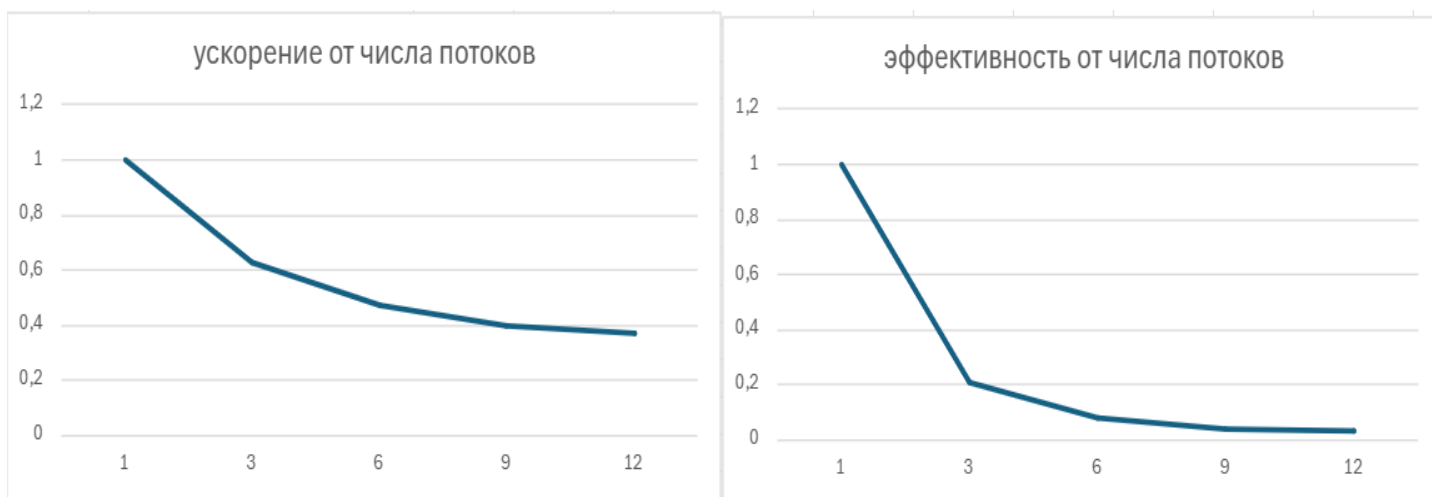
После полученных значений и обработки на то, что они введены корректно создается нужное количество потоков для обработки каждого фильтра. Изначально созданная матрица копируется в `result`.

### Суть метода свёртки:

1. Формируется окно вокруг нынешнего элемента.
2. Из окна берутся все элементы.
3. Находится сумма чисел окна.
4. Сумма делится на кол-во элементов в этом окне
5. Центральное значение элемента заменяется на найденное значение

Ниже приведены данные, показывающие изменения ускорения и эффективности, с разным количеством потоков, для этой реализации.

| Число потоков | Время выполнения | Ускорение | Эффективность |
|---------------|------------------|-----------|---------------|
| 1             | 120              | 1,00      | 1,00          |
| 3             | 190              | 0,63      | 0,21          |
| 6             | 256              | 0,47      | 0,08          |
| 9             | 295              | 0,40      | 0,04          |
| 12            | 325              | 0,37      | 0,03          |



```

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <unistd.h>
#include <time.h>

#define MAX_MATRIX_SIZE 1000

int num_threads;
int matrix_size;
int filter_iterations;
double **matrix;
double **result;
int filter_size;

pthread_mutex_t lock; // Мьютекс для синхронизации потоков

void *apply_filter(void *arg) {
    int thread_id = *(int *)arg;
    int chunk_size = matrix_size / num_threads;
    int start = thread_id * chunk_size;
    int end = (thread_id == num_threads - 1) ? matrix_size : start + chunk_size;
    for (int iter = 0; iter < filter_iterations; iter++) {
        for (int i = start; i < end; i++) {
            for (int j = 0; j < matrix_size; j++) {
                double sum = 0.0;
                for (int fi = -filter_size / 2; fi <= filter_size / 2; fi++) {
                    for (int fj = -filter_size / 2; fj <= filter_size / 2; fj++) {
                        int ni = i + fi, nj = j + fj;
                        if (ni >= 0 && ni < matrix_size && nj >= 0 && nj < matrix_size) {
                            sum += matrix[ni][nj];
                        }
                    }
                }
                int size_lenght;
                int size_high;
            }
        }
    }
}

```

```

                if (i < filter_size / 2) {
                    size_lenght = filter_size / 2 + i + 1;
                }
                else if (matrix_size - i - 1 < filter_size / 2) {
                    size_lenght = filter_size / 2 + matrix_size - i;
                }
                else {
                    size_lenght = filter_size;
                }
                if (j < filter_size / 2) {
                    size_high = filter_size / 2 + j + 1;
                }
                else if (matrix_size - j - 1 < filter_size / 2) {
                    size_high = filter_size / 2 + matrix_size - j;
                }
                else {
                    size_high = filter_size;
                }
                result[i][j] = sum / (size_lenght * size_high);
            }
        }
        pthread_mutex_lock(&lock);
        double **temp = matrix;
        matrix = result;
        result = temp;
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}

void initialize_matrix() {
    matrix = malloc(matrix_size * sizeof(double *));
    result = malloc(matrix_size * sizeof(double *));
}

```

```

        for (int i = 0; i < matrix_size; i++) {
            matrix[i] = malloc(matrix_size * sizeof(double));
            result[i] = malloc(matrix_size * sizeof(double));
            for (int j = 0; j < matrix_size; j++) {
                matrix[i][j] = rand() % 100;
                result[i][j] = 0.0;
            }
        }
    }

void free_matrix() {
    for (int i = 0; i < matrix_size; i++) {
        free(matrix[i]);
        free(result[i]);
    }
    free(matrix);
    free(result);
}

void print_matrix(double **mat, const char *title) {
    printf("%s:\n", title);
    for (int i = 0; i < matrix_size; i++) {
        for (int j = 0; j < matrix_size; j++) {
            printf("%.2f ", mat[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

int main(int argc, char *argv[]) {
    if (argc != 5) {
        printf("Использование: %s <размер матрицы> <размер фильтра> <итераций фильтра> <количество потоков>\n", argv[0]);
        return 1;
    }
    matrix_size = atoi(argv[1]);
}

```

```

filter_size = atoi(argv[2]);
if (filter_size % 2 == 0) {
    printf("размер фильтра должен быть нечётным");
    return 1;
}
filter_iterations = atoi(argv[3]);
num_threads = atoi(argv[4]);
if (matrix_size <= 0 || filter_size <= 0 || filter_iterations <= 0 || num_threads <= 0) {
    printf("Все параметры должны быть положительными целыми числами.\n");
    return 1;
}
if (matrix_size > MAX_MATRIX_SIZE) {
    printf("Размер матрицы превышает максимально допустимый (%d).\n", MAX_MATRIX_SIZE);
    return 1;
}
initialize_matrix();
print_matrix(matrix, "Начальная матрица");
pthread_t threads[num_threads];
int thread_ids[num_threads];
pthread_mutex_init(&lock, NULL);
clock_t start_time = clock();
for (int i = 0; i < num_threads; i++) {
    thread_ids[i] = 1;
    if (pthread_create(&threads[i], NULL, apply_filter, &thread_ids[i]) != 0) {
        perror("pthread_create");
        return 1;
    }
}
for (int i = 0; i < num_threads; i++) {
    pthread_join(threads[i], NULL);
}
clock_t end_time = clock();
double time_spent = (double)(end_time - start_time) / 1;
pthread_mutex_destroy(&lock);
print_matrix(matrix, "Фильтрованная матрица");
printf("Время, затраченное на фильтрацию: %.6f секунд\n", time_spent);

```

```

free_matrix();
return 0;
}

```

Начальная матрица:

```

83.00 86.00 77.00
15.00 93.00 35.00
86.00 92.00 49.00

```

Фильтрованная матрица:

```

69.25 64.83 72.75
75.83 68.44 72.00
71.50 61.67 67.25

```

Время, затраченное на фильтрацию: 0.000095 секунд

Начальная матрица:

```

83.00 86.00 77.00 15.00 93.00
35.00 86.00 92.00 49.00 21.00
62.00 27.00 90.00 59.00 63.00
26.00 40.00 26.00 72.00 36.00
11.00 68.00 67.00 29.00 82.00

```

Фильтрованная матрица:

```

66.91 66.25 62.95 58.62 55.46
62.68 62.70 60.90 58.06 55.71
54.46 55.63 56.52 56.24 55.38
47.25 49.34 52.53 54.67 55.24
43.78 46.21 50.39 53.53 54.75

```

Время, затраченное на фильтрацию: 0.000120 секунд

tulchinskij@LAPTOP-QIG5MTAH:~\$ ./lab 5 5 5 2

Начальная матрица:

```

83.00 86.00 77.00 15.00 93.00
35.00 86.00 92.00 49.00 21.00
62.00 27.00 90.00 59.00 63.00
26.00 40.00 26.00 72.00 36.00
11.00 68.00 67.00 29.00 82.00

```

Фильтрованная матрица:

```

28.65 28.75 29.24 29.51 29.72
21.49 21.56 21.93 22.14 22.29
26.02 26.13 26.49 26.71 26.89
23.30 23.44 23.69 23.92 24.04
18.75 18.80 18.84 18.89 18.93

```

Время, затраченное на фильтрацию: 0.000149 секунд

# Протокол работы программы

```
execve("./lab", ["/lab", "5", "5", "1", "1"], 0x7ffd4050b530 /* 35 vars */) = 0
brk(NULL) = 0x557d97f8c000
arch_prctl(0x3001 /* ARCH_??? */ , 0x7ffc09d6b8c0) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f559274b000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=16995, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 16995, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f5592746000
close(3) = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0"... , 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"... , 784, 64) = 784
pread64(3, "\4\0\0\0\0\0\0\0\5\0\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0"... , 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\17\357\204\3$\f\221\2039x\324\224\323\236S"... , 68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"... , 784, 64) = 784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f559251d000
mprotect(0x7f5592545000, 2023424, PROT_NONE) = 0
mmap(0x7f5592545000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f5592545000
mmap(0x7f55926da000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f55926da000
mmap(0x7f5592733000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f5592733000
mmap(0x7f5592739000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f5592739000
close(3) = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f559251a000
arch_prctl(ARCH_SET_FS, 0x7f559251a740) = 0
set_tid_address(0x7f559251aa10) = 31673
set_robust_list(0x7f559251aa20, 24) = 0
rseq(0x7f559251b0e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f5592733000, 16384, PROT_READ) = 0
mprotect(0x557d91f8c000, 4096, PROT_READ) = 0
mprotect(0x7f5592785000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f5592746000, 16995) = 0
getrandom("\xbfx25\xb0\xdc\xec\x0d\x90\x3b", 8, GRND_NONBLOCK) = 8
brk(NULL) = 0x557d97f8c000
brk(0x557d97fad000) = 0x557d97fad000
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x4), ...}, AT_EMPTY_PATH) = 0
write(1, "\320\235\320\260\321\207\320\260\320\273\321\214\320\275\320\260\321\217\320\274\320\260\321\202\321\200\320\270\321\206\320"... , 35Начальная матрица:
) = 35
write(1, "83.00 86.00 77.00 15.00 93.00 \n", 3183.00 86.00 77.00 15.00 93.00
) = 31
write(1, "35.00 86.00 92.00 49.00 21.00 \n", 3135.00 86.00 92.00 49.00 21.00
) = 31
write(1, "62.00 27.00 90.00 59.00 63.00 \n", 3162.00 27.00 90.00 59.00 63.00
) = 31
write(1, "26.00 40.00 26.00 72.00 36.00 \n", 3126.00 40.00 26.00 72.00 36.00
) = 31
write(1, "11.00 68.00 67.00 29.00 82.00 \n", 3111.00 68.00 67.00 29.00 82.00
) = 31
write(1, "\n", 1
) = 1
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=5770200}) = 0
rt_sigaction(SIGRT_1, {sa_handler=0x7f55925ae870, sa_mask=[], sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f559255f520}, NULL, 8) = 0
rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f5591d19000
mprotect(0x7f5591d1a000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f5592519910, parent_tid=0x7f5592519910, exit_signal=0, stack=0x7f5591d19000, stack_size=0x7fff00, tls=0x7f5592519640} => {parent_tid=[31674]}, 88) = 31674
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=6714400}) = 0
write(1, "\320\244\320\270\320\273\321\214\321\202\321\200\320\276\320\262\320\260\320\275\320\275\320\260\321\217\320\274\320\260\321"... , 43Фильтрованная матрица:
) = 43
write(1, "70.89 63.42 62.53 63.17 62.11 \n", 3170.89 63.42 62.53 63.17 62.11
) = 31
```

```

write(1, "60.83 57.81 56.90 58.25 57.75 \n", 3160.83 57.81 56.90 58.25 57.75) = 31write(1, "58.40 55.00 55.80
58.90 58.07 \n", 3158.40 55.00 55.80 58.90 58.07) = 31
write(1, "52.50 52.44 52.05 56.69 57.17 \n", 3152.50 52.44 52.05 56.69 57.17
) = 31
write(1, "46.33 48.08 50.53 54.92 58.22 \n", 3146.33 48.08 50.53 54.92 58.22
) = 31
write(1, "\n", 1
) = 1
write(1, "\320\222\321\200\320\265\320\274\321\217, \320\267\320\260\321\202\321\200\320\260\321\207\320
\265\320\275\320\275\320\276"..., 84
,
: 0.000944
) = 84
exit_group(0) = ?
+++ exited with 0 +++

```

## Вывод

В ходе написания данной лабораторной работы я научился создавать программы, работающие с несколькими потоками, а также синхронизировать их между собой. В результате тестирования программы, я проанализировал каким образом количество потоков влияет на эффективность и ускорение работы программы. Оказалось, что большое количество потоков даёт хорошее ускорение на больших количествах входных данных, но эффективность использования ресурсов находится на приемлемом уровне только на небольшом количестве потоков, не превышающем количества логических ядер процессора. Лабораторная работа была довольно интересна, так как я впервые работал с многопоточностью и синхронизацией на СИ.