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-210Б-23

Студент: Абдыкалыков Н. А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 14.02.25

Москва, 2025

Постановка задачи

Вариант 12.

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

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

Общий метод и алгоритм решения

Использованные системные вызовы:

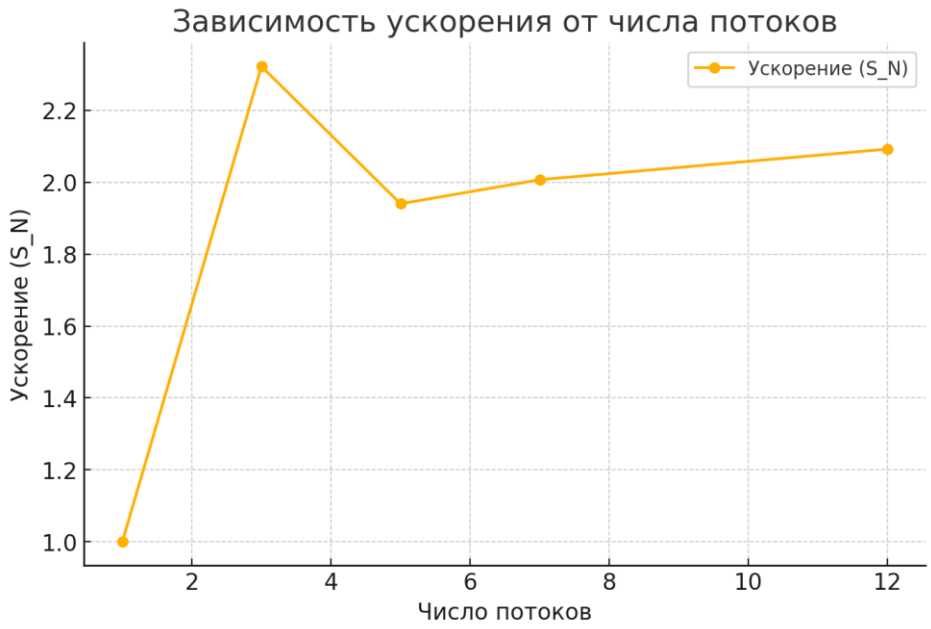
- ssize_t read(int fd, void *buf, size_t count); – читает данные из файлового дескриптора
- ssize_t write(int fd, const void *buf, size_t count); – записывает count байт из буфера buf в файловый дескриптор fd
- int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), void *arg); – создаёт новый поток
- int pthread_join(pthread_t thread, void **retval); – ожидает завершения потока
- int clock_gettime(clockid_t clk_id, struct timespec *tp); – получает текущее значение времени

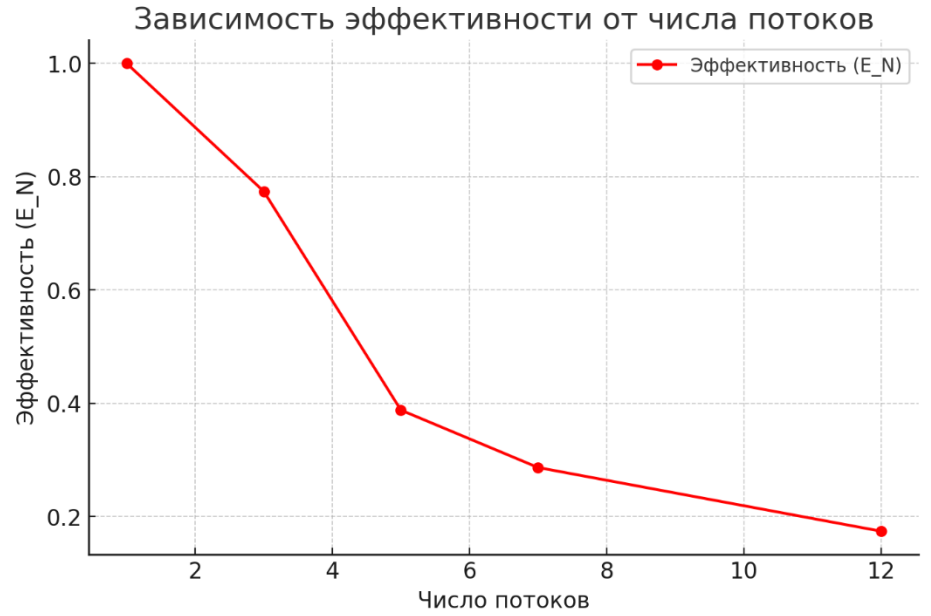
Программа выполняет параллельную обработку матрицы, применяя к ней K раз фильтры эрозии и наращивания. Количество потоков, число итераций K, а также размеры матрицы передаются через аргументы командной строки.

Матрица считывается построчно из стандартного ввода, при этом каждая строка должна содержать ровно cols вещественных чисел. После загрузки данных программа распределяет вычисления между потоками. По завершении всех итераций программа ожидает завершения потоков, выводит результирующие матрицы и фиксирует время выполнения. Для синхронизации используется pthread_join, а замер времени осуществляется с помощью clock_gettime.

Программа была протестирована на входных данных: матрица 200 на 200 из случайных элементов, 5 итераций.

Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	19.646	1.00	1.00
3	8.460	2.32	0.77
5	10.126	1.94	0.39
7	9.789	2.01	0.29
12	9.390	2.09	0.17





Результаты показывают, что увеличение числа потоков сначала ускоряет выполнение, но после трёх потоков эффективность снижается. Это связано с накладными расходами на управление потоками и конкуренцией за ресурсы. Оптимальное распределение достигается при 3-5 потоках, после чего прирост производительности становится незначительным.

Код программы

main.c

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

```
#define BUFFER_SIZE 4096

void print_str(const char* s) { write(STDOUT_FILENO, s, strlen(s)); }

void print_error(const char* s) { write(STDERR_FILENO, s, strlen(s)); }

int read_line(char* buf, size_t size) {
	size_t i = 0;
	char c;
	ssize_t n;
	while (i < size - 1) {
		n = read(STDIN_FILENO, &c, 1);
		if (n < 0) return -1;  // ошибка чтения
		if (n == 0)            // EOF
			break;
		if (c == '\n') break;
		buf[i++] = c;
	}
	buf[i] = '\0';
	return (int)i;
}

void print_matrix(const char* title, double* m, int rows, int cols) {
	write(STDOUT_FILENO, title, strlen(title));
	write(STDOUT_FILENO, "\n", 1);
	char buf[128];
	for (int i = 0; i < rows; i++) {
		for (int j = 0; j < cols; j++) {
			int len = snprintf(buf, sizeof(buf), "%.2f ", m[i * cols + j]);
			if (len > 0) write(STDOUT_FILENO, buf, len);
		}
		write(STDOUT_FILENO, "\n", 1);
	}
}
```

```
typedef struct {
	int start_row;
	int end_row;
	int rows;
	int cols;
	int filter;  // 0 – эрозия, 1 – наращивание
	double* in;
	double* out;
} ThreadArgs;

void* compute_segment(void* arg) {
	ThreadArgs* args = (ThreadArgs*)arg;
	int rstart = args->start_row;
	int rend = args->end_row;
	int cols = args->cols;
	int total_rows = args->rows;
	for (int i = rstart; i < rend; i++) {
		for (int j = 0; j < cols; j++) {
			double result = args->in[i * cols + j];
			if (args->filter == 0) {
				// Эрозия: ищем минимальное значение
				for (int di = -1; di <= 1; di++) {
					for (int dj = -1; dj <= 1; dj++) {
						int ni = i + di;
						int nj = j + dj;
						if (ni >= 0 && ni < total_rows && nj >= 0 && nj
< cols) {
							double val = args->in[ni * cols + nj];
							if (val < result) result = val;
						}
					}
				}
			} else {
				// Наращивание: ищем максимальное значение
				for (int di = -1; di <= 1; di++) {
```

```
                                for (int dj = -1; dj <= 1; dj++) {
                                        int ni = i + di;
                                        int nj = j + dj;
                                        if (ni >= 0 && ni < total_rows && nj >= 0 && nj
< cols) {
                                                double val = args->in[ni * cols + nj];
                                                if (val > result) result = val;
                                        }
                                }
                        }
                }
                args->out[i * cols + j] = result;
        }
    }
    return NULL;
}

int parse_double(const char* token, double* value) {
    char* endptr;
    *value = strtod(token, &endptr);
    return (endptr != token);
}

int main(int argc, char* argv[]) {
    if (argc < 5) {
        print_str("usage: program <max_threads> <K> <rows> <cols>\n");
        return 1;
    }

    const int max_threads = atoi(argv[1]);
    const int K = atoi(argv[2]);
    const int rows = atoi(argv[3]);
    const int cols = atoi(argv[4]);

    if (max_threads <= 0 || K <= 0 || rows <= 0 || cols <= 0) {
        print_str("Неправильно заданы параметры\n");
```

```
	return 1;
}

// Выделение памяти под исходную матрицу
double* matrix = malloc(rows * cols * sizeof(double));
if (!matrix) {
	print_str("Ошибка выделения памяти\n");
	return 1;
}

// Построчный ввод: читаем rows строк, каждая должна содержать ровно cols чисел
char line[BUFFER_SIZE];
for (int r = 0; r < rows; r++) {
	if (read_line(line, sizeof(line)) < 0) {
		perror("read_line");
		free(matrix);
		return 1;
	}
	int count = 0;
	char* saveptr;
	char* token = strtok_r(line, " \t", &saveptr);
	while (token != NULL) {
		if (count >= cols) {
			print_str("Введено слишком много столбцов\n");
			free(matrix);
			return 1;
		}
		if (!parse_double(token, &matrix[r * cols + count])) {
			print_error("Неправильное число в строке\n");
			free(matrix);
			return 1;
		}
		count++;
		token = strtok_r(NULL, " \t", &saveptr);
	}
```

```
	if (count != cols) {
		print_str("Введено слишком мало столбцов\n");
		free(matrix);
		return 1;
	}
}

// Выделение памяти под рабочие матрицы для эрозии и наращивания
double* erosion_in = malloc(rows * cols * sizeof(double));
double* erosion_out = malloc(rows * cols * sizeof(double));
double* dilation_in = malloc(rows * cols * sizeof(double));
double* dilation_out = malloc(rows * cols * sizeof(double));
if (!erosion_in || !erosion_out || !dilation_in || !dilation_out) {
	print_str("Ошибка выделения памяти\n");
	free(matrix);
	free(erosion_in);
	free(erosion_out);
	free(dilation_in);
	free(dilation_out);
	return 1;
}
memcpy(erosion_in, matrix, rows * cols * sizeof(double));
memcpy(dilation_in, matrix, rows * cols * sizeof(double));
free(matrix);

// Определяем число потоков: не больше max_threads и не больше числа строк
int num_threads = (max_threads < rows) ? max_threads : rows;
pthread_t threads[num_threads];
ThreadArgs args[num_threads];

// Замер времени исполнения
struct timespec start_time, end_time;
if (clock_gettime(CLOCK_MONOTONIC, &start_time) == -1) {
	perror("clock_gettime");
	return 1;
```

```
	}

	// Эрозия: K итераций
	for (int iter = 0; iter < K; iter++) {
		int rows_per_thread = rows / num_threads;
		int remainder = rows % num_threads;
		int current_row = 0;
		for (int i = 0; i < num_threads; i++) {
			args[i].start_row = current_row;
			int extra = (i < remainder) ? 1 : 0;
			args[i].end_row = current_row + rows_per_thread + extra;
			current_row = args[i].end_row;
			args[i].rows = rows;
			args[i].cols = cols;
			args[i].filter = 0;  // эрозия
			args[i].in = erosion_in;
			args[i].out = erosion_out;
			if (pthread_create(&threads[i], NULL, compute_segment, &args[i]) !=
0) {
				perror("pthread_create");
				free(erosion_in);
				free(erosion_out);
				free(dilation_in);
				free(dilation_out);
				return 1;
			}
		}
		for (int i = 0; i < num_threads; i++) {
			pthread_join(threads[i], NULL);
		}
		// Обмен указателей для следующей итерации
		double* temp = erosion_in;
		erosion_in = erosion_out;
		erosion_out = temp;
	}
	// Итоговый результат эрозии находится в erosion_in
```

```
	// Наращивание: K итераций
	for (int iter = 0; iter < K; iter++) {
		int rows_per_thread = rows / num_threads;
		int remainder = rows % num_threads;
		int current_row = 0;
		for (int i = 0; i < num_threads; i++) {
			args[i].start_row = current_row;
			int extra = (i < remainder) ? 1 : 0;
			args[i].end_row = current_row + rows_per_thread + extra;
			current_row = args[i].end_row;
			args[i].rows = rows;
			args[i].cols = cols;
			args[i].filter = 1;  // наращивание
			args[i].in = dilation_in;
			args[i].out = dilation_out;
			if (pthread_create(&threads[i], NULL, compute_segment, &args[i]) !=
0) {
				perror("pthread_create");
				free(erosion_in);
				free(erosion_out);
				free(dilation_in);
				free(dilation_out);
				return 1;
			}
		}
		for (int i = 0; i < num_threads; i++) {
			pthread_join(threads[i], NULL);
		}
		double* temp = dilation_in;
		dilation_in = dilation_out;
		dilation_out = temp;
	}
	// Итоговый результат наращивания находится в dilation_in

	if (clock_gettime(CLOCK_MONOTONIC, &end_time) == -1) {
```

```
            perror("clock_gettime");
            return 1;
        }
        double elapsed_sec = end_time.tv_sec - start_time.tv_sec;
        double elapsed_nsec = end_time.tv_nsec - start_time.tv_nsec;
        if (elapsed_nsec < 0) {
            elapsed_sec -= 1;
            elapsed_nsec += 1000000000;
        }

        print_matrix("Результат эрозии:", erosion_in, rows, cols);
        print_matrix("Результат наращивания:", dilation_in, rows, cols);

        char time_msg[128];
        int len = snprintf(time_msg, sizeof(time_msg), "Затрачено %lf секунд\n",
                           elapsed_sec + elapsed_nsec / 1e9);
        if (len > 0) write(STDOUT_FILENO, time_msg, len);

        free(erosion_in);
        free(erosion_out);
        free(dilation_in);
        free(dilation_out);
        return 0;
}
```

Протокол работы программы

Тестирование:

```
$ ./lab_2 1 5 10 10 < m_1.txt
Результат эрозии:
-9.26 -9.26 -9.26 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75
-9.26 -9.26 -9.26 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75
-9.26 -9.26 -9.26 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75
-9.26 -9.26 -9.26 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75
-9.26 -9.26 -9.26 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75
-9.26 -9.26 -9.26 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75
-9.26 -9.26 -9.26 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75
```

```
-9.26 -9.26 -9.26 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75
-9.26 -9.26 -9.26 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75
-9.24 -9.24 -9.24 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75 -9.75
Результат наращивания:
9.59 9.59 9.91 9.91 9.91 9.91 9.91 9.91 9.91 9.91
9.59 9.59 9.91 9.91 9.91 9.91 9.91 9.91 9.91 9.91
9.59 9.59 9.91 9.91 9.91 9.91 9.91 9.91 9.91 9.91
9.59 9.59 9.91 9.93 9.93 9.93 9.93 9.93 9.93 9.93
9.59 9.59 9.91 9.93 9.93 9.93 9.93 9.93 9.93 9.93
9.59 9.59 9.91 9.93 9.93 9.93 9.93 9.93 9.93 9.93
9.59 9.59 9.91 9.93 9.93 9.93 9.93 9.93 9.93 9.93
9.59 9.59 9.91 9.93 9.93 9.93 9.93 9.93 9.93 9.93
9.24 9.24 9.64 9.93 9.93 9.93 9.93 9.93 9.93 9.93
8.98 8.98 9.64 9.93 9.93 9.93 9.93 9.93 9.93 9.93
Затрачено 0.001092 секунд
$ ./lab_2 4 5 10 10 < m_1.txt | tail -n 1
Затрачено 0.002796 секунд
$ ./lab_2 1 5 100 100 < m_2.txt | tail -n 1
Затрачено 0.005825 секунд
$ ./lab_2 4 5 100 100 < m_2.txt | tail -n 1
Затрачено 0.003094 секунд
```

Strace:

```
$ strace ./lab_2 4 1 10 10 < m_1.txt
execve("./lab_2", ["./lab_2", "4", "1", "10", "10"], 0x7ffd80592100 /* 21 vars */) = 0
brk(NULL)                               = 0x5564e5b1b000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fd89957e000
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
fstat(3, {st_mode=S_IFREG|0644, st_size=20723, ...}) = 0
mmap(NULL, 20723, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd899578000
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\0\3\0>\0\1\0\0\0\220\243\2\0\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

     fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 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, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd899366000

     mmap(0x7fd89938e000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fd89938e000

     mmap(0x7fd899516000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7fd899516000

     mmap(0x7fd899565000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1fe000) = 0x7fd899565000

     mmap(0x7fd89956b000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7fd89956b000

     close(3)                                = 0

     mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fd899363000

     arch_prctl(ARCH_SET_FS, 0x7fd899363740) = 0

     set_tid_address(0x7fd899363a10)         = 879230

     set_robust_list(0x7fd899363a20, 24)     = 0

     rseq(0x7fd899364060, 0x20, 0, 0x53053053) = 0

     mprotect(0x7fd899565000, 16384, PROT_READ) = 0

     mprotect(0x5564bb550000, 4096, PROT_READ) = 0

     mprotect(0x7fd8995b6000, 8192, PROT_READ) = 0

     prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

     munmap(0x7fd899578000, 20723)           = 0

     getrandom("\x13\x47\xb5\x7e\x3e\x2b\x13\x70", 8, GRND_NONBLOCK) = 8

     brk(NULL)                               = 0x5564e5b1b000

     brk(0x5564e5b3c000)                     = 0x5564e5b3c000

     read(0, "-", 1)                         = 1

     read(0, "2", 1)                         = 1

     read(0, ".", 1)                         = 1

     ...

     read(0, "\n", 1)                        = 1

     rt_sigaction(SIGRT_1, {sa_handler=0x7fd8993ff530, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7fd8993ab330}, 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) =
0x7fd898b62000
```

```
mprotect(0x7fd898b63000, 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=0x7fd899362990, parent_tid=0x7fd899362990, exit_signal=0, stack=0x7fd898b62000, stack_size=0x7fff80, tls=0x7fd8993626c0} => {parent_tid=[879231]}, 88) = 879231

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7fd898361000

mprotect(0x7fd898362000, 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=0x7fd898b61990, parent_tid=0x7fd898b61990, exit_signal=0, stack=0x7fd898361000, stack_size=0x7fff80, tls=0x7fd898b616c0} => {parent_tid=[879232]}, 88) = 879232

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7fd897b60000

mprotect(0x7fd897b61000, 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=0x7fd898360990, parent_tid=0x7fd898360990, exit_signal=0, stack=0x7fd897b60000, stack_size=0x7fff80, tls=0x7fd8983606c0} => {parent_tid=[879233]}, 88) = 879233

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7fd89735f000

mprotect(0x7fd897360000, 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=0x7fd897b5f990, parent_tid=0x7fd897b5f990, exit_signal=0, stack=0x7fd89735f000, stack_size=0x7fff80, tls=0x7fd897b5f6c0} => {parent_tid=[0]}, 88) = 879234

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 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=0x7fd897b5f990, parent_tid=0x7fd897b5f990, exit_signal=0, stack=0x7fd89735f000, stack_size=0x7fff80, tls=0x7fd897b5f6c0} => {parent_tid=[879235]}, 88) = 879235

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 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=0x7fd898360990, parent_tid=0x7fd898360990, exit_signal=0, stack=0x7fd897b60000, stack_size=0x7fff80, tls=0x7fd8983606c0} => {parent_tid=[0]}, 88) = 879236
```

```
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|C
LONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7fd898b61990,
parent_tid=0x7fd898b61990, exit_signal=0, stack=0x7fd898361000, stack_size=0x7fff80,
tls=0x7fd898b616c0} => {parent_tid=[0]}, 88) = 879237

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|C
LONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7fd899362990,
parent_tid=0x7fd899362990, exit_signal=0, stack=0x7fd898b62000, stack_size=0x7fff80,
tls=0x7fd8993626c0} => {parent_tid=[0]}, 88) = 879238

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202
\321\215\321\200\320\276\320\267\320\270\320\270:", 32) = 32

write(1, "\n", 1)                       = 1

write(1, "-3.74 ", 6)                   = 6

write(1, "-3.74 ", 6)                   = 6

write(1, "-3.74 ", 6)                   = 6

...

write(1, "9.93 ", 5)                    = 5

write(1, "9.93 ", 5)                    = 5

write(1, "9.93 ", 5)                    = 5

write(1, "\n", 1)                       = 1

write(1, "\320\227\320\260\321\202\321\200\320\260\321\207\320\265\320\275\320\276
0.015660 \321\201\320\265"..., 41) = 41

exit_group(0)                           = ?

+++ exited with 0 +++
```

Вывод

В ходе выполнения лабораторной работы я освоил работу с потоками в POSIX и научился эффективно распределять вычисления между ними. В процессе тестирования обнаружил, что при большом количестве потоков эффективность падает из-за накладных расходов. Также возникла проблема с некорректным разбиением матрицы при неравномерном распределении строк, что приводило к неверным вычислениям.