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:

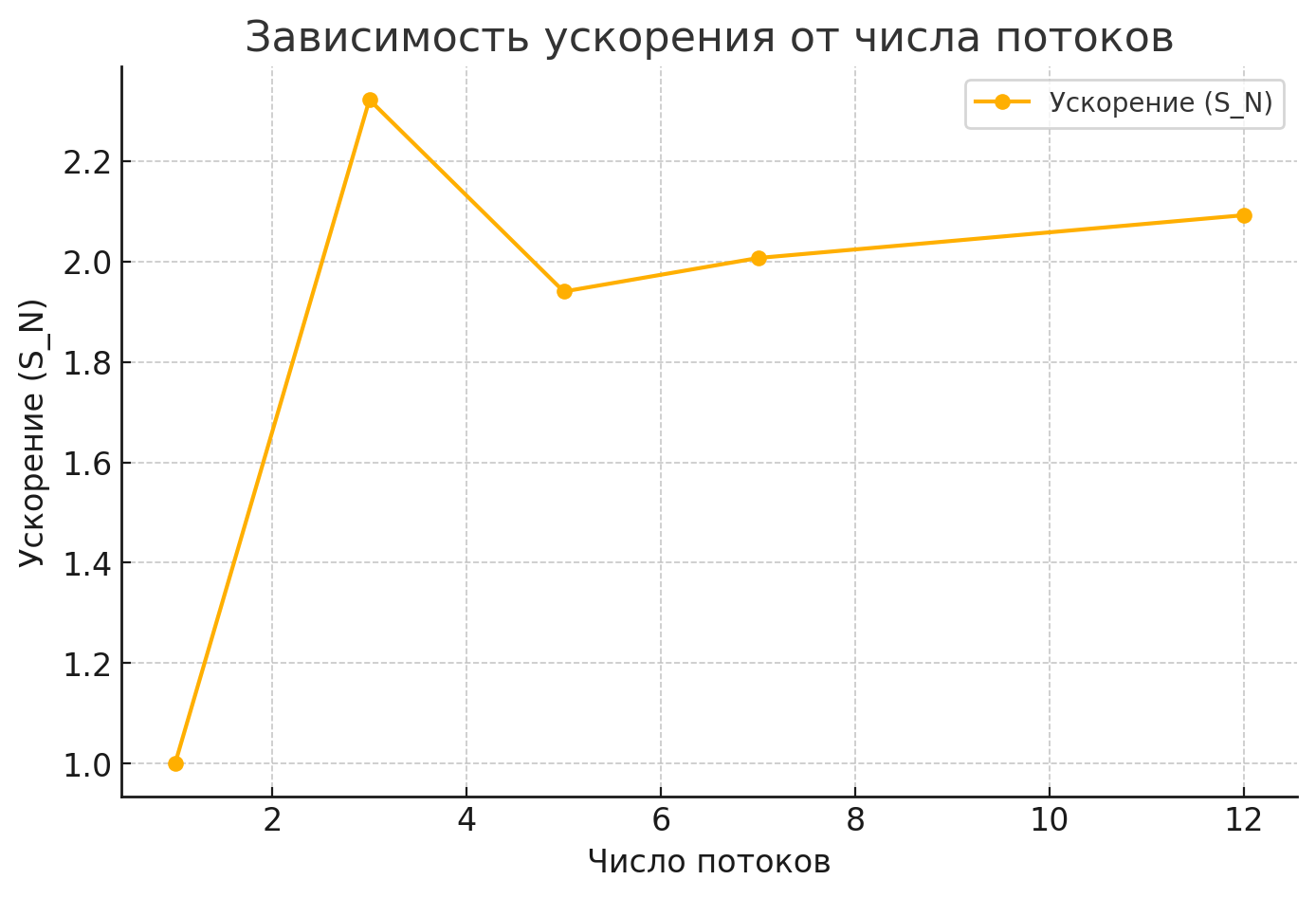
* ssize\_t read(int fd, void \*buf, size\_t count); – читает данные из файлового дескриптора
* ssize\_t write(int fd, const void \*buf, size\_t count); – записывает count байт из буфера buf в файловый дескриптор fd
* int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start\_routine)(void \*), 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

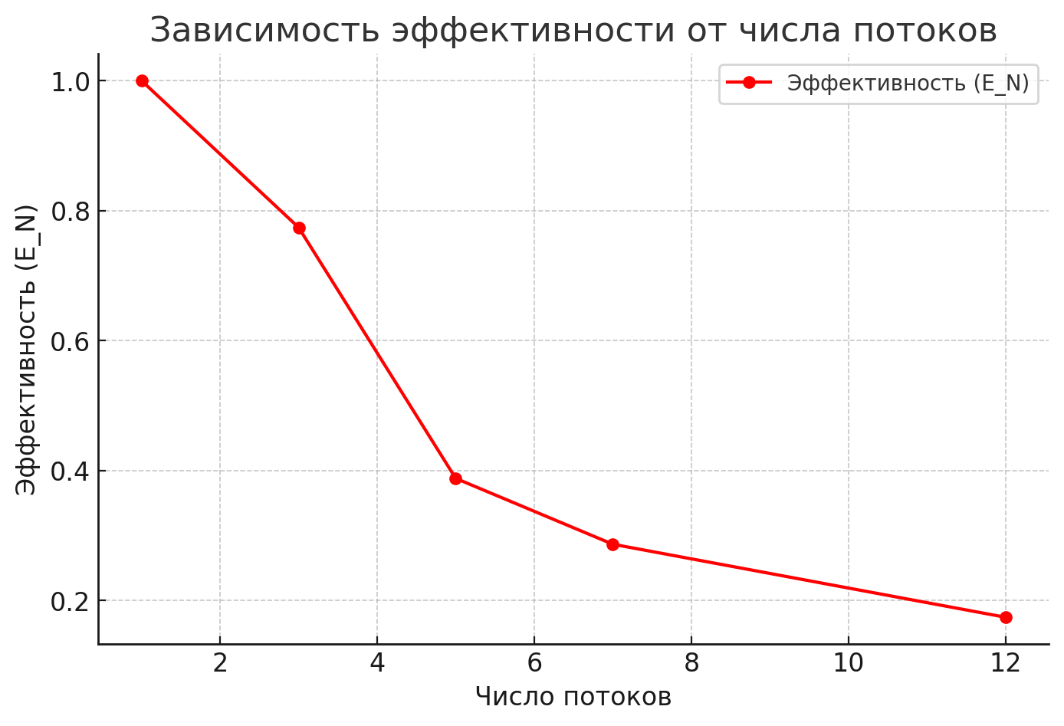
Программа выполняет параллельную обработку матрицы, применяя к ней K раз фильтры эрозии и наращивания. Количество потоков, число итераций K, а также размеры матрицы передаются через аргументы командной строки.

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|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=[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|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=[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.