

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: Нугаев М. Э.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 03.01.25

Москва, 2025

Постановка задачи

Вариант 16.

Цель работы

Приобретение практических навыков в:

- Управление процессами 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.

Задается радиус окружности. Необходимо с помощью метода Монте-Карло рассчитать её площадь.

Общий метод и алгоритм решения

Использованные системные вызовы:

- `ssize_t write(int fd, const void *buf, size_t count);` - записывает `count` байт из буфера в файл.
- `int pthread_create(pthread_t *thread, const pthread_attr_t *attr,`
`void *(*start)(void *), void *arg)` – создание потока
- `int pthread_join (pthread_t THREAD_ID, void ** DATA)` – ожидание завершения потока
- `int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr)` – инициализация мьютекса
- `int pthread_mutex_lock(pthread_mutex_t *mutex)` – блокировка мьютекса
- `int pthread_mutex_unlock(pthread_mutex_t *mutex)` – разблокировка мьютекса
- `int pthread_mutex_destroy(pthread_mutex_t *mutex)` – удаление мьютекса

Аргументам командной строки пользователь должен ввести 3 аргумента: радиус окружности, количество точек для метода Монте-Карло, количество потоков. Программа валидирует вводимые пользователем данные. Создается динамический массив указателей на потоки и структуры, в которых содержатся поля: `long *inside_points` – количество попавших внутрь точек, `pthread_mutex_t *mutex` - указатель на мьютекс, который отвечает за добавление счетчика точек, попавших внутрь круга, каждого потока в общий счётчик., `long radius` – радиус окружности, `long numPoints_per_Thread` – количество точек на поток; данную структуру мы будем использовать при подсчете точек попавших внутрь окружности. Каждый N-ый поток отвечает за проверку 1/N от количества всех точек по методом Монте-Карло. В начале итерации поток проверяет каждую из случайно сделанных внутри квадрата точек на попадание внутрь окружности. В конце поток блокирует мьютекс, чтобы добавить локальный счетчик точек внутри

круга к общему счетчику точек внутри круга, после чего разблокирует мьютекс. После завершения всех потоков, рассчитывается площадь по формуле и в `STDOUT_FILENO` выводится площадь окружности.

Количество потоков	Время, с	Ускорение	Эффективность
1	2,912	1	1
2	1,495	1,947826087	0,973913043
3	1,225	2,377142857	0,792380952
4	0,769	3,786736021	0,946684005
5	0,656	4,43902439	0,887804878
6	0,594	4,902356902	0,817059484
7	0,510	5,709803922	0,815686275
8	0,466	6,248927039	0,78111588
9	0,405	7,190123457	0,798902606
10	0,369	7,891598916	0,789159892
11	0,343	8,489795918	0,771799629
12	0,327	8,905198777	0,742099898

Данные подсчитаны при количестве точек = 214483647

Код программы

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <sys/wait.h>
#include <limits.h>
#include <pthread.h>
#include <float.h>

#define MAX_THREADS 33

void write_error(const char *error_string)
{
    if (error_string == NULL)
    {
        write(STDERR_FILENO, "ERROR\n", 7);
    }
    write(STDERR_FILENO, error_string, strlen(error_string));
}

// будем использовать структуру, тк pthread_create принимает только один аргумент typedef struct
{
    long *inside_points; pthread_mutex_t
    *mutex; long radius;
```

```

    long numPoints_per_Thread;

} THREADS_POINTS;

typedef enum Errors
{
    E_SUCCESS = 0,
    E_INVALID_INPUT,
    E_NOT_ENOUGH_PARAMS,
    E_INVALID_ARG,
    E_TYPE_OVERFLOW,
    E_MEMORY_ALLOCATION,
    E_CANNOT_OPEN_FILE,
    E_INVALID_EPSILON,
    E_DEREFENCE_NULL_POINTER,
    E_INT_OVERFLOW,
    E_LONG_OVERFLOW,
    E_FLOAT_OVERFLOW,
    E_DOUBLE_OVERFLOW,
    E_LONG_DOUBLE_OVERFLOW,
    E_INVALID_FLAG_ARG,
    E_SAME_FILE_NAMES,
    E_BUFFER_OVERFLOW,
    E_SAME_FILES,
    E_FALSE,
} ERRORS_EXIT_CODES;

/* Перевод строки в int */
ERRORS_EXIT_CODES string_to_long_int(const char *str_number, long *int_result_number, int base) {
    if (str_number == NULL || int_result_number == NULL)
        return E_INVALID_INPUT;    char *endptr;
    *int_result_number = strtol(str_number, &endptr, base);
    if (*int_result_number == LONG_MAX || *int_result_number == LONG_MIN)    return
E_TYPE_OVERFLOW;    else if (*endptr != '\0')    return E_INVALID_INPUT;    return E_SUCCESS;
}

/* Проверка на переполнение типа double */ int
is_double_overflow(float value)
{
    if (value > DBL_MAX || value < -DBL_MAX || value == HUGE_VALF || value == -
HUGE_VALF)
    {
        return 1;
    }
    if (isinf(value))
    {
        return 1;
    }
    return 0;
}

/* Перевод строки в double */
ERRORS_EXIT_CODES string_to_double(const char *str, double *num)
{
    if (str == NULL || num == NULL)    return
E_DEREFENCE_NULL_POINTER;

    char *endptr;
    double value = strtod(str, &endptr);    if (*endptr != '\0')
return E_INVALID_INPUT;

```

```

    if (is_double_overflow(value))    return
    E_DOUBLE_OVERFLOW;

    *num = value;    return
    E_SUCCESS;
}

// Функция для расчёта площади методом Монте-Карло
void *monteCarloCircleArea(void *arg)
{
    double x, y, radius;    long pointsInCircle
    = 0;
    THREADS_POINTS *thread_points = (THREADS_POINTS *)arg;
    radius = thread_points->radius;
    long numPoints = thread_points->numPoints_per_Thread;

    /* rand() - cringe, очень медленно считает, поэтому юзаем rand_r*/    time_t seed = time(NULL);
    for (long i = 0; i < numPoints; i++)
    {
        // Генерируем случайные x и y в диапазоне [-R, R]    x = ((double)rand_r(&seed) /
        RAND_MAX) * 2 * radius - radius;    y = ((double)rand_r(&seed) / RAND_MAX) * 2 * radius - radius;
        // Проверяем, попадает ли точка в окружность    if (x * x + y * y <=
        radius * radius)
        {
            pointsInCircle++;
        }
    }
    pthread_mutex_lock(thread_points->mutex);
    *(thread_points->inside_points) += pointsInCircle;    pthread_mutex_unlock(thread_points->mutex);

    return NULL;
}

void write_result(const char *result)
{
    if (result == NULL)    return;

    if (write(STDOUT_FILENO, result, strlen(result)) == -1)    exit(EXIT_FAILURE);
}

int main(int argc, char *argv[])
{
    if (argc != 4 || argv == NULL || argv[1] == NULL || argv[2] == NULL || argv[3] ==
    NULL)
    {
        write_error("ERROR: INVALID INPUT, must be 1)radius 2)number of points 3)number of threads\n");
        return E_INVALID_INPUT;
    }
    double radius;    long
    numPoints;    long numThreads;
    /* Корректно обрабатываем числа*/
    ERRORS_EXIT_CODES error = string_to_double(argv[1], &radius);    if (error ==
    E_DOUBLE_OVERFLOW)
    {
        write_error("ERROR_DOUBLE_OVERFLOW\n");    return error;
    }
    else if (error == E_INVALID_INPUT)
    {
        write_error("INVALID INPUT IN FILE\n");    return error;
    }
}

```

```

else if (error != E_SUCCESS)
{
    write_error("ERROR\n");    return error;
}
error = string_to_long_int(argv[2], &numPoints, 10);
if (error == E_TYPE_OVERFLOW)
{
    write_error("ERROR_INT_OVERFLOW\n");    return error;
}
else if (error == E_INVALID_INPUT)
{
    write_error("INVALID INPUT IN FILE\n");    return error;
}
else if (error != E_SUCCESS)
{
    write_error("ERROR\n");    return error;
}
error = string_to_long_int(argv[3], &numThreads, 10);
if (error == E_TYPE_OVERFLOW)
{
    write_error("ERROR_INT_OVERFLOW\n");    return error;
}
else if (error == E_INVALID_INPUT)
{
    write_error("INVALID INPUT IN FILE\n");    return error;
}
else if (error != E_SUCCESS)
{
    write_error("ERROR\n");    return error;
}
if (radius < 0 || numPoints <= 0 || numThreads > MAX_THREADS)
{
    write_error("ERROR: INVALID_INPUT\n");    return E_INVALID_INPUT;
}
pthread_t *threads = (pthread_t *)malloc(numThreads * sizeof(pthread_t));
if (threads == NULL)
{
    write_error("ERROR: MEMORY ALLOCATION\n");    return
E_MEMORY_ALLOCATION;
}
THREADS_POINTS *thread_points = (THREADS_POINTS *)malloc(numThreads *
sizeof(THREADS_POINTS)); if (thread_points == NULL)
{
    free(threads);
    write_error("ERROR: MEMORY ALLOCATION\n");    return
E_MEMORY_ALLOCATION;
}
long numPoints_per_Thread = numPoints / numThreads;
long pointsInCircle = 0;

pthread_mutex_t mutex;
if (pthread_mutex_init(&mutex, NULL) != 0)
{
    free(threads);    free(thread_points);
    write_error("ERROR: MUTEX INIT\n");    return
E_MEMORY_ALLOCATION;
}
for (long i = 0; i < numThreads; i++)
{

```

```

    thread_points[i].radius = radius;
    thread_points[i].numPoints_per_Thread = numPoints_per_Thread;    thread_points[i].inside_points =
&pointsInCircle;    thread_points[i].mutex = &mutex;
}
for (long i = 0; i < numThreads; i++)
{
    pthread_create(threads + i, NULL, monteCarloCircleArea, thread_points + i); }
for (long i = 0; i < numThreads; i++)
{
    pthread_join(threads[i], NULL);
}
pthread_mutex_destroy(&mutex);

double circle_area = (double)pointsInCircle / numPoints * 4 * radius * radius;
char result[BUFSIZ];
if (sprintf(result, "Square = %lf\n", circle_area) < 0)
{
    free(threads);    free(thread_points);
    write_error("ERROR: BUFFER OVERFLOW\n");
    return E_BUFFER_OVERFLOW;
} write_result(result);
free(threads);    free(thread_points);

    return E_SUCCESS; }

```

Протокол работы программы

miro@DESKTOP-GD72A05/LABS/lab2/src\$ time ./main 1 214483647 1 Square = 3.141596

real 0m2.912s user 0m2.902s sys 0m0.010s miro@DESKTOP-

GD72A05/OS_LABS/lab2/src\$ time ./main 1 214483647 2 Square = 3.141621

real 0m1.495s user 0m2.977s sys 0m0.000s miro@DESKTOP-

GD72A05/OS_LABS/lab2/src\$ time ./main 1 214483647 3 Square = 3.141582

real 0m1.225s user 0m3.437s sys 0m0.000s miro@DESKTOP-

GD72A05/OS_LABS/lab2/src\$ time ./main 1 214483647 4 Square = 3.141581

real 0m0.769s user 0m3.026s sys 0m0.000s miro@DESKTOP-

GD72A05/OS_LABS/lab2/src\$ time ./main 1 214483647 5 Square = 3.141677

real 0m0.656s user 0m3.165s sys 0m0.011s miro@DESKTOP-

GD72A05/OS_LABS/lab2/src\$ time ./main 1 214483647 6 Square = 3.141170

real 0m0.594s user 0m3.328s sys 0m0.001s miro@DESKTOP-

GD72A05/OS_LABS/lab2/src\$ time ./main 1 214483647 7 Square = 3.141364

real 0m0.510s user 0m3.340s sys 0m0.001s miro@DESKTOP-

GD72A05/OS_LABS/lab2/src\$ time ./main 1 214483647 8 Square = 3.141457

real 0m0.466s user 0m3.573s sys 0m0.001s miro@DESKTOP-

GD72A05/OS_LABS/lab2/src\$ time ./main 1 214483647 9 Square = 3.141867

real 0m0.405s user 0m3.517s sys 0m0.001s miro@DESKTOP-

GD72A05/OS_LABS/lab2/src\$ time ./main 1 214483647 10 Square = 3.141412

real 0m0.369s user 0m3.584s sys 0m0.001s miro@DESKTOP-

GD72A05/OS_LABS/lab2/src\$ time ./main 1 214483647 11 Square = 3.141717

real 0m0.343s user 0m3.671s sys 0m0.001s miron@DESKTOP-

GD72A05/OS_LABS/lab2/src\$ time ./main 1 214483647 12 Square = 3.141007

real 0m0.327s user 0m3.789s sys 0m0.001s miron@DESKTOP-

GD72A05/OS_LABS/lab2/src\$ time ./main 1 214483647 15 Square = 3.142074

real 0m0.404s user 0m3.694s sys 0m0.010s miron@DESKTOP-

GD72A05/OS_LABS/lab2/src\$ time ./main 1 214483647 20 Square = 3.142080

real 0m0.361s user 0m3.719s sys 0m0.001s miron@DESKTOP-

GD72A05/OS_LABS/lab2/src\$ time ./main 1 214483647 30 Square = 3.141784

real 0m0.330s

user 0m3.624s

sys 0m0.010s

Strace:

```
miron@DESKTOP-GD72A05/LABS/lab2/src$ strace ./main 1 9000099 12 execve("./main",  
[ "./main", "1", "9000099", "12"], 0x7ffcac765838 /* 30 vars */) = 0 brk(NULL)  
= 0x558ce7d4e000 arch_prctl(0x3001 /* ARCH_??? */, 0x7fffac12d2b0) = -1 EINVAL  
(Invalid argument)  
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =  
0x7effd36a9000 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=30263, ...}, AT_EMPTY_PATH) = 0 mmap(NULL, 30263, PROT_READ,  
MAP_PRIVATE, 3, 0) = 0x7effd36a1000 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"..., 784, 64) = 784 pread64(3, "\4\0\0\0  
\0\0\0\5\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\0I\17\357\204\3$\f221\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"..., 784, 64) = 784  
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7effd3478000  
mprotect(0x7effd34a0000, 2023424, PROT_NONE) = 0  
mmap(0x7effd34a0000, 1658880, PROT_READ|PROT_EXEC,  
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7effd34a0000  
mmap(0x7effd3635000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,  
0x1bd000) = 0x7effd3635000  
mmap(0x7effd368e000, 24576, PROT_READ|PROT_WRITE,
```

```

MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7effd368e000
mmap(0x7effd3694000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7effd3694000 close(3)
= 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7effd3475000 arch_prctl(ARCH_SET_FS, 0x7effd3475740) = 0 set_tid_address(0x7effd3475a10)
= 30999 set_robust_list(0x7effd3475a20, 24) = 0 rseq(0x7effd34760e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7effd368e000, 16384, PROT_READ) = 0 mprotect(0x558ce6a0e000, 4096,
PROT_READ) = 0 mprotect(0x7effd36e3000, 8192, PROT_READ) = 0 prlimit64(0,
RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7effd36a1000, 30263) = 0 getrandom("\x1e\x76\xd7\xf4\x9e\x40\xb9\x59", 8,
GRND_NONBLOCK) = 8 brk(NULL) = 0x558ce7d4e000 brk(0x558ce7d6f000)
= 0x558ce7d6f000
rt_sigaction(SIGRT_1, {sa_handler=0x7effd3509870, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,
sa_restorer=0x7effd34ba520}, 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) =
0x7effd2c74000 mprotect(0x7effd2c75000, 8388608,
PROT_READ|PROT_WRITE) = 0 rt_sigprocmask(SIG_BLOCK, ~[],
[], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7effd3474910, parent_tid=0x7effd3474910, exit_signal=0,
stack=0x7effd2c74000, stack_size=0x7fff00, tls=0x7effd3474640} => {parent_tid=[31000]}, 88) = 31000
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7effd2473000 mprotect(0x7effd2474000, 8388608,
PROT_READ|PROT_WRITE) = 0 rt_sigprocmask(SIG_BLOCK, ~[],
[], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7effd2c73910, parent_tid=0x7effd2c73910, exit_signal=0,
stack=0x7effd2473000, stack_size=0x7fff00, tls=0x7effd2c73640} => {parent_tid=[31001]}, 88) = 31001
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7effd1c72000 mprotect(0x7effd1c73000, 8388608,
PROT_READ|PROT_WRITE) = 0 rt_sigprocmask(SIG_BLOCK, ~[],
[], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7effd2472910, parent_tid=0x7effd2472910, exit_signal=0,
stack=0x7effd1c72000, stack_size=0x7fff00, tls=0x7effd2472640} => {parent_tid=[31002]}, 88) = 31002
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7effd1471000 mprotect(0x7effd1472000, 8388608,
PROT_READ|PROT_WRITE) = 0 rt_sigprocmask(SIG_BLOCK, ~[],
[], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,

```

```

child_tid=0x7effd1c71910, parent_tid=0x7effd1c71910, exit_signal=0,
stack=0x7effd1471000, stack_size=0x7fff00, tls=0x7effd1c71640} => {parent_tid=[31003]}, 88) = 31003
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7effd0c70000 mprotect(0x7effd0c71000, 8388608,
PROT_READ|PROT_WRITE) = 0 rt_sigprocmask(SIG_BLOCK, ~[],
[], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7effd1470910, parent_tid=0x7effd1470910, exit_signal=0,
stack=0x7effd0c70000, stack_size=0x7fff00, tls=0x7effd1470640} => {parent_tid=[31004]}, 88) = 31004
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7effd046f000 mprotect(0x7effd0470000, 8388608,
PROT_READ|PROT_WRITE) = 0 rt_sigprocmask(SIG_BLOCK, ~[],
[], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7effd0c6f910, parent_tid=0x7effd0c6f910, exit_signal=0,
stack=0x7effd046f000, stack_size=0x7fff00, tls=0x7effd0c6f640} => {parent_tid=[31005]}, 88) = 31005
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7effcfc6e000 mprotect(0x7effcfc6f000, 8388608,
PROT_READ|PROT_WRITE) = 0 rt_sigprocmask(SIG_BLOCK, ~[],
[], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_
SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|C
LONG_CHILD_CLEARTID,
child_tid=0x7effd046e910, parent_tid=0x7effd046e910, exit_signal=0,
stack=0x7effcfc6e000, stack_size=0x7fff00, tls=0x7effd046e640} => {parent_tid=[31006]}, 88) = 31006
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7effcf46d000 mprotect(0x7effcf46e000, 8388608,
PROT_READ|PROT_WRITE) = 0 rt_sigprocmask(SIG_BLOCK, ~[],
[], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7effcfc6d910, parent_tid=0x7effcfc6d910, exit_signal=0,
stack=0x7effcf46d000, stack_size=0x7fff00, tls=0x7effcfc6d640} => {parent_tid=[31007]}, 88) = 31007
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7effcec6c000 mprotect(0x7effcec6d000, 8388608,
PROT_READ|PROT_WRITE) = 0 rt_sigprocmask(SIG_BLOCK, ~[],
[], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7effcf46c910, parent_tid=0x7effcf46c910, exit_signal=0,
stack=0x7effcec6c000, stack_size=0x7fff00, tls=0x7effcf46c640} => {parent_tid=[31008]}, 88) = 31008

```

```

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7effce46b000 mprotect(0x7effce46c000, 8388608,
PROT_READ|PROT_WRITE) = 0 rt_sigprocmask(SIG_BLOCK, ~[],
[], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7effcec6b910, parent_tid=0x7effcec6b910,      exit_signal=0,
stack=0x7effce46b000, stack_size=0x7fff00, tls=0x7effcec6b640} => {parent_tid=[31009]}, 88) = 31009
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7effcdc6a000 mprotect(0x7effcdc6b000, 8388608,
PROT_READ|PROT_WRITE) = 0 rt_sigprocmask(SIG_BLOCK, ~[],
[], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7effce46a910, parent_tid=0x7effce46a910,      exit_signal=0,
stack=0x7effcdc6a000, stack_size=0x7fff00, tls=0x7effce46a640} => {parent_tid=[31010]}, 88) = 31010
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7effcd469000 mprotect(0x7effcd46a000, 8388608,
PROT_READ|PROT_WRITE) = 0 rt_sigprocmask(SIG_BLOCK, ~[],
[], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7effcdc69910, parent_tid=0x7effcdc69910,      exit_signal=0,
stack=0x7effcd469000, stack_size=0x7fff00, tls=0x7effcdc69640} => {parent_tid=[31011]}, 88) = 31011
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
futex(0x7effd3474910, Futex_WAIT_BITSET|Futex_CLOCK_REALTIME, 31000, NULL,
Futex_BITSET_MATCH_ANY) = 0
futex(0x7effd2c73910, Futex_WAIT_BITSET|Futex_CLOCK_REALTIME, 31001, NULL,
Futex_BITSET_MATCH_ANY) = 0
futex(0x7effd2472910, Futex_WAIT_BITSET|Futex_CLOCK_REALTIME, 31002, NULL,
Futex_BITSET_MATCH_ANY) = 0
futex(0x7effd1470910, Futex_WAIT_BITSET|Futex_CLOCK_REALTIME, 31004, NULL,
Futex_BITSET_MATCH_ANY) = 0
munmap(0x7effd2c74000, 8392704) = 0
futex(0x7effd0c6f910, Futex_WAIT_BITSET|Futex_CLOCK_REALTIME, 31005, NULL,
Futex_BITSET_MATCH_ANY) = 0
munmap(0x7effd2473000, 8392704) = 0 munmap(0x7effd1c72000,
8392704) = 0
futex(0x7effcf6d910, Futex_WAIT_BITSET|Futex_CLOCK_REALTIME, 31007, NULL,
Futex_BITSET_MATCH_ANY) = -1 EAGAIN (Resource temporarily unavailable)
munmap(0x7effd1471000, 8392704) = 0
futex(0x7effcf46c910, Futex_WAIT_BITSET|Futex_CLOCK_REALTIME, 31008, NULL,
Futex_BITSET_MATCH_ANY) = 0
munmap(0x7effd0c70000, 8392704) = 0 munmap(0x7effd046f000,
8392704) = 0

```

```
futex(0x7effce46a910, Futex_WAIT_BITSET|Futex_CLOCK_REALTIME, 31010, NULL,
Futex_BITSET_MATCH_ANY) = 0
munmap(0x7effcf6e000, 8392704) = 0
futex(0x7effcdc69910, Futex_WAIT_BITSET|Futex_CLOCK_REALTIME, 31011, NULL,
Futex_BITSET_MATCH_ANY) = 0
munmap(0x7effcf46d000, 8392704) = 0 write(1,
"Square = 3.139363\n", 18Square = 3.139363
) = 18 exit_group(0)
= ?
+++ exited with 0 +++
```

Вывод

В ходе лабораторной работы приобретены навыки управления потоками в ОС и их синхронизации с использованием стандартных средств pthread для Unix. Разработана многопоточная программа для расчёта площади круга методом Монте-Карло, с возможностью ограничения количества одновременно работающих потоков. Реализация успешно продемонстрировала корректность вычислений и эффективность синхронизации, а также важность грамотного управления потоками для оптимизации работы программы.