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: Дворников М.Д.
Преподаватель: Бахарев В.Д.
Оценка: _____
Дата: 18.12.24

Москва, 2024

Постановка задачи

Цель работы

Приобретение практических навыков в:

- Управление потоками в ОС
- Обеспечение синхронизации между потоками

Задание

Составить программу на языке Си(C++), 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.

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.

6 вариант:

Произвести перемножение 2-ух матриц, содержащих комплексные числа. Произвести перемножение 2-ух матриц, содержащих комплексные числа

Общий метод и алгоритм решения

Использованные системные вызовы:

- int pthread_create(pthread_t *restrict newthread, const pthread_attr_t *restrict attr, void *(*start_routine)(void *), void *restrict arg)
  Создаёт поток, выполняющий указанную функцию (start_routine) с заданными аргументами (arg).
- int pthread_join(pthread_t th, void **thread_return)
  Ожидает завершения указанного потока и получает результат его выполнения.
- ssize_t write(int fd, const void *buf, size_t n)
  Записывает n байт из буфера buf в файл с файловым дескриптором fd. Возвращает количество записанных байт или -1 в случае ошибки.
- void exit(int status)
  Завершает выполнение программы, закрывая все потоки и освобождая ресурсы.
- Для реализации с использованием atomic:
  Библиотека <stdatomic.h>:
  - Тип данных _Atomic double для представления атомарных значений действительных и мнимых частей матриц.
  - Макросы atomic_store(PTR, VAL) для безопасной записи значения и atomic_load(PTR) для безопасного чтения.
- Для реализации с использованием mutex:
  - pthread_mutex_t — тип данных для работы с мьютексами.
  - int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr)
    Инициализация мьютекса.
  - int pthread_mutex_lock(pthread_mutex_t *mutex)
    Блокирует мьютекс, ограничивая доступ других потоков к защищаемому ресурсу.

- int pthread_mutex_unlock(pthread_mutex_t *mutex)
  Разблокирует мьютекс, разрешая доступ другим потокам.
- int pthread_mutex_destroy(pthread_mutex_t *mutex)
  Уничтожает мьютекс, освобождая связанные с ним ресурсы.

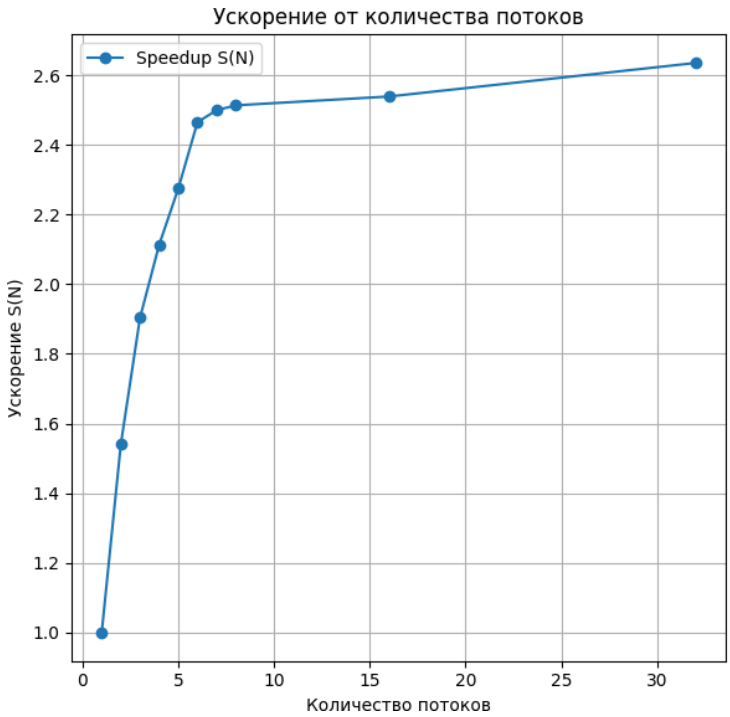
Матрицы генерируются случайным образом. Элементы матриц — комплексные числа, где действительная и мнимая части находятся в диапазоне от -10 до 10. Для генерации случайных чисел используется rand() и диапазон задаётся константой RAND_RANGE.
В программе создаются потоки в количестве, указанном в аргументе командной строки. Каждый поток обрабатывает определённое количество строк итоговой матрицы.
Для реализации с использованием атомиков:
Итоговая матрица представлена как массив атомарных структур, содержащих действительную и мнимую части.
Потоки безопасно записывают значения в итоговую матрицу с помощью операций atomic_store.
Для реализации с использованием мьютексов:
Потоки защищают доступ к итоговой матрице с помощью мьютекса.
Каждая запись в итоговую матрицу выполняется внутри защищённого блока между вызовами pthread_mutex_lock и pthread_mutex_unlock.
После завершения работы всех потоков родительский поток объединяет результаты и выводит итоговую матрицу.
Синхронизация потоков осуществляется с использованием атомиков или мьютексов для предотвращения состояния гонки и обеспечения корректности вычислений.

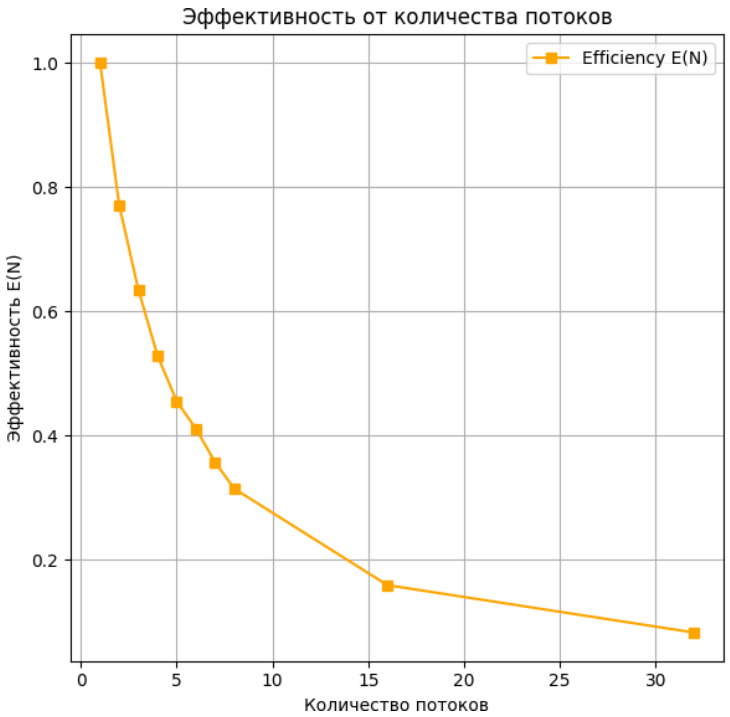
```
matrix_size = 1000
```

Число потоков	Время выполнения, с	Ускорение	Эффективность
1	20.148	1,00	1,00
2	13.073	1.54	0.77
3	10.584	1.90	0,923
4	9.535	2.11	0,895
5	8.858	2.27	0,816
6	8.174	2.46	0,748
7	8.062	2.50	0,69
8	8.017	2.51	0,639
16	7.937	2.54	0,586
32	7.647	2.64	0,545

Ускорение $S_N=T_1/T_N$:
Например, для 2 потоков:
$S_2=20.148/13.073\approx1.54$

Эффективность $E_N=S_N/N$:
Например, для 2 потоков:
$E_2=1.54/2\approx0.77$





1. Ускорение S_N от количества потоков:
Ускорение S_N увеличивается с ростом числа потоков **N**, но этот рост постепенно замедляется. На начальных этапах (до 4-8 потоков) ускорение растет довольно быстро, а затем приближается к плато.
Рост ускорения ограничивается законом Амдала. Чем больше потоков используется, тем меньше времени уходит на параллельные задачи. Однако накладные расходы на управление потоками (координация, синхронизация) и доля последовательного кода начинают преобладать, что ограничивает максимальное ускорение.
При увеличении потоков ускорение растет, но не пропорционально числу потоков, а рост становится всё менее эффективным.

2. Эффективность E_N от количества потоков:
Эффективность E_N значительно падает при увеличении количества потоков. На малом числе потоков эффективность близка к 1 (100%), а при большем количестве потоков она снижается до 0.2−0.30.2-0.30.2−0.3 и ниже.
Эффективность показывает, насколько хорошо используются ресурсы. При большем числе потоков накладные расходы на синхронизацию и управление потоками начинают занимать большую долю времени, а прирост производительности от каждого дополнительного потока уменьшается. Кроме того, при увеличении потоков могут возникать конфликты за ресурсы (например, доступ к памяти), что дополнительно снижает эффективность.
Максимальная эффективность достигается при оптимальном соотношении количества потоков к характеру задачи. При избыточном числе потоков эффективность снижается.

Таким образом, мы подтвердили действие закона Амдала: параллельные вычисления имеют предел, после которого добавление потоков перестает давать значительный прирост производительности.

Для данной задачи оптимальное количество потоков находится в диапазоне от 4 до 8, когда ускорение близко к линейному, а эффективность всё ещё достаточно высока.
Увеличение потоков сверх этого диапазона приводит к существенному снижению эффективности, хотя ускорение продолжает расти, но с минимальной отдачей.

Код программы

mutex.c

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

#define MAX_THREADS 32
#define RAND_RANGE 10

typedef double complex cplx;

pthread_mutex_t mutex;
cplx **result;

typedef struct {
    size_t start_row;
    size_t end_row;
    size_t size;
    cplx **matrix1;
    cplx **matrix2;
} ThreadArgs;

void HandleError(const char *msg) {
    write( fd: STDERR_FILENO,  buf: msg,  nbyte: strlen( s: msg));
    exit(EXIT_FAILURE);
}

void AllocateMatrix(cplx ***matrix, size_t size) {
    *matrix = malloc( size: size * sizeof(cplx *));
    if (*matrix == NULL) {
        HandleError( msg: "Ошибка выделения памяти для матрицы.\n");
    }
    for (size_t i = 0; i < size; i++) {
        (*matrix)[i] = malloc( size: size * sizeof(cplx));
        if ((*matrix)[i] == NULL) {
            HandleError( msg: "Ошибка выделения памяти для строки матрицы.\n");
        }
    }
}

void FreeMatrix(cplx **matrix, size_t size) {
    for (size_t i = 0; i < size; i++) {
        free(matrix[i]);
    }
    free(matrix);
}

cplx GenerateRandomComplex() {
    double real = (rand() % (2 * RAND_RANGE + 1)) - RAND_RANGE;
    double imag = (rand() % (2 * RAND_RANGE + 1)) - RAND_RANGE;
    return real + imag * I;
}

void *MatrixMultiply(void *args) {
    ThreadArgs *data = (ThreadArgs *)args;

    for (size_t i = data->start_row; i < data->end_row; i++) {
        for (size_t j = 0; j < data->size; j++) {
            cplx sum = 0;
            for (size_t k = 0; k < data->size; k++) {
                sum += data->matrix1[i][k] * data->matrix2[k][j];
            }

            // Синхронизация доступа к результату
            if (pthread_mutex_lock(&mutex) != 0) {
                HandleError( msg: "Ошибка блокировки мьютекса.\n");
            }
            result[i][j] = sum;
            if (pthread_mutex_unlock(&mutex) != 0) {
                HandleError( msg: "Ошибка разблокировки мьютекса.\n");
            }
        }
    }

    return NULL;
}

int main(int argc, char **argv) {
    if (argc != 3) {
        HandleError( msg: "Использование: ./mutex <количество потоков> <размер матрицы>\n");
    }

    size_t threads_count = strtoul( str: argv[1],  endptr: NULL,  base: 10);
    if (threads_count > MAX_THREADS) {
        HandleError( msg: "Ошибка: Превышено максимальное количество потоков.\n");
    }

    size_t matrix_size = strtoul( str: argv[2],  endptr: NULL,  base: 10);

    cplx **matrix1, **matrix2;

    AllocateMatrix( matrix: &matrix1,  size: matrix_size);
    AllocateMatrix( matrix: &matrix2,  size: matrix_size);
    AllocateMatrix( matrix: &result,  size: matrix_size);

    srand(time(NULL));

    for (size_t i = 0; i < matrix_size; i++) {
        for (size_t j = 0; j < matrix_size; j++) {
            matrix1[i][j] = GenerateRandomComplex();
            matrix2[i][j] = GenerateRandomComplex();
        }
    }

    if (pthread_mutex_init(&mutex, NULL) != 0) {
        HandleError( msg: "Ошибка инициализации мьютекса.\n");
    }

    pthread_t threads[MAX_THREADS];
    ThreadArgs thread_args[MAX_THREADS];

    size_t rows_per_thread = matrix_size / threads_count;
    for (size_t i = 0; i < threads_count; i++) {
        thread_args[i].start_row = i * rows_per_thread;
        thread_args[i].end_row = (i == threads_count - 1) ? matrix_size : (i + 1) * rows_per_thread;
        thread_args[i].size = matrix_size;
        thread_args[i].matrix1 = matrix1;
        thread_args[i].matrix2 = matrix2;

        if (pthread_create(&threads[i], NULL, MatrixMultiply, &thread_args[i]) != 0) {
            HandleError( msg: "Ошибка создания потока.\n");
        }
    }

    for (size_t i = 0; i < threads_count; i++) {
        if (pthread_join(threads[i], NULL) != 0) {
            HandleError( msg: "Ошибка ожидания потока.\n");
        }
    }

    pthread_mutex_destroy(&mutex);

    for (size_t i = 0; i < matrix_size; i++) {
        for (size_t j = 0; j < matrix_size; j++) {
            char buffer[64];
            snprintf(buffer, sizeof(buffer), "(%.2f + %.2fi) ", creal(result[i][j]), cimag(result[i][j]));
            write( fd: STDOUT_FILENO,  buf: buffer,  nbyte: strlen( s: buffer));
        }
        write( fd: STDOUT_FILENO,  buf: "\n",  nbyte: 1);
    }

    FreeMatrix( matrix: matrix1,  size: matrix_size);
    FreeMatrix( matrix: matrix2,  size: matrix_size);
    FreeMatrix( matrix: result,  size: matrix_size);

    return EXIT_SUCCESS;
}
```

atomic.c

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

#define MAX_THREADS
#define RAND_RANGE 10

typedef double complex cplx;

typedef struct {
    _Atomic double real;
    _Atomic double imag;
} atomic_cplx;

atomic_cplx **atomic_result;

typedef struct {
    size_t start_row;
    size_t end_row;
    size_t size;
    cplx **matrix1;
    cplx **matrix2;
} ThreadArgs;

void HandleError(const char *msg) {
    write( fd: STDERR_FILENO,  buf: msg,  nbyte: strlen( s: msg));
    exit(EXIT_FAILURE);
}

void AllocateMatrix(cplx ***matrix, size_t size) {
    *matrix = malloc( size: size * sizeof(cplx *));
    if (*matrix == NULL) {
        HandleError( msg: "Ошибка выделения памяти для матрицы.\n");
    }
    for (size_t i = 0; i < size; i++) {
        (*matrix)[i] = malloc( size: size * sizeof(cplx));
        if ((*matrix)[i] == NULL) {
            HandleError( msg: "Ошибка выделения памяти для строки матрицы.\n");
        }
    }
}

void FreeMatrix(cplx **matrix, size_t size) {
    for (size_t i = 0; i < size; i++) {
        free(matrix[i]);
    }
    free(matrix);
}

void AllocateAtomicMatrix(atomic_cplx ***matrix, size_t size) {
    *matrix = malloc( size: size * sizeof(atomic_cplx *));
    if (*matrix == NULL) {
        HandleError( msg: "Ошибка выделения памяти для атомарной матрицы.\n");
    }
    for (size_t i = 0; i < size; i++) {
        (*matrix)[i] = malloc( size: size * sizeof(atomic_cplx));
        if ((*matrix)[i] == NULL) {
            HandleError( msg: "Ошибка выделения памяти для строки атомарной матрицы.\n");
        }
    }
}

void FreeAtomicMatrix(atomic_cplx **matrix, size_t size) {
    for (size_t i = 0; i < size; i++) {
        free(matrix[i]);
    }
    free(matrix);
}

void *MatrixMultiply(void *args) {
    ThreadArgs *data = (ThreadArgs *)args;

    for (size_t i = data->start_row; i < data->end_row; i++) {
        for (size_t j = 0; j < data->size; j++) {
            cplx sum = 0;
            for (size_t k = 0; k < data->size; k++) {
                sum += data->matrix1[i][k] * data->matrix2[k][j];
            }

            atomic_store(&atomic_result[i][j].real, creal(sum));
            atomic_store(&atomic_result[i][j].imag, cimag(sum));
        }
    }

    return NULL;
}

cplx GenerateRandomComplex() {
    double real = (rand() % (2 * RAND_RANGE + 1)) - RAND_RANGE;
    double imag = (rand() % (2 * RAND_RANGE + 1)) - RAND_RANGE;
    return real + imag * I;
}

int main(int argc, char **argv) {
    if (argc != 3) {
        HandleError( msg: "Использование: ./atomic <количество потоков> <размер матрицы>\n");
    }

    size_t threads_count = strtoul( str: argv[1],  endptr: NULL,  base: 10);
    if (threads_count > MAX_THREADS) {
        HandleError( msg: "Ошибка: Превышено максимальное количество потоков.\n");
    }

    size_t matrix_size = strtoul( str: argv[2],  endptr: NULL,  base: 10);

    cplx **matrix1, **matrix2;

    AllocateMatrix( matrix: &matrix1,  size: matrix_size);
    AllocateMatrix( matrix: &matrix2,  size: matrix_size);
    AllocateAtomicMatrix( matrix: &atomic_result,  size: matrix_size);

    srand(time(NULL));

    for (size_t i = 0; i < matrix_size; i++) {
        for (size_t j = 0; j < matrix_size; j++) {
            matrix1[i][j] = GenerateRandomComplex();
            matrix2[i][j] = GenerateRandomComplex();
        }
    }

    pthread_t threads[MAX_THREADS];
    ThreadArgs thread_args[MAX_THREADS];

    size_t rows_per_thread = matrix_size / threads_count;
    for (size_t i = 0; i < threads_count; i++) {
        thread_args[i].start_row = i * rows_per_thread;
        thread_args[i].end_row = (i == threads_count - 1) ? matrix_size : (i + 1) * rows_per_thread;
        thread_args[i].size = matrix_size;
        thread_args[i].matrix1 = matrix1;
        thread_args[i].matrix2 = matrix2;

        if (pthread_create(&threads[i], NULL, MatrixMultiply, &thread_args[i]) != 0) {
            HandleError( msg: "Ошибка создания потока.\n");
        }
    }

    for (size_t i = 0; i < threads_count; i++) {
        if (pthread_join(threads[i], NULL) != 0) {
            HandleError( msg: "Ошибка ожидания потока.\n");
        }
    }

    for (size_t i = 0; i < matrix_size; i++) {
        for (size_t j = 0; j < matrix_size; j++) {
            double real = atomic_load(&atomic_result[i][j].real);
            double imag = atomic_load(&atomic_result[i][j].imag);

            char buffer[64];
            snprintf(buffer, sizeof(buffer), "(%.2f + %.2fi) ", real, imag);
            write( fd: STDOUT_FILENO,  buf: buffer,  nbyte: strlen( s: buffer));
        }
        write( fd: STDOUT_FILENO,  buf: "\n",  nbyte: 1);
    }

    FreeMatrix( matrix: matrix1,  size: matrix_size);
    FreeMatrix( matrix: matrix2,  size: matrix_size);
    FreeAtomicMatrix( matrix: atomic_result,  size: matrix_size);

    return EXIT_SUCCESS;
}
```

Протокол работы программы

Тестирование:

```
./mutex 1 1000  15.63s user 0.49s system 78% cpu 20.620 total
./mutex 2 1000  18.18s user 0.49s system 132% cpu 14.134 total
./mutex 8 1000  21.24s user 0.51s system 273% cpu 7.944 total
./mutex 16 1000  18.82s user 0.51s system 251% cpu 7.686 total
```

dtrace:

```
❯ sudo dtruss ./mutex 2 20
SYSCALL(args)          = return
munmap(0x101008000, 0x84000)           = 0 0
munmap(0x10108C000, 0x8000)            = 0 0
munmap(0x101094000, 0x4000)            = 0 0
munmap(0x101098000, 0x4000)            = 0 0
munmap(0x10109C000, 0x48000)           = 0 0
munmap(0x1010E4000, 0x4C000)           = 0 0
crossarch_trap(0x0, 0x0, 0x0)          = -1 Err#45
open(".\0", 0x100000, 0x0)             = 3 0
fcntl(0x3, 0x32, 0x16EFC70E8)          = 0 0
close(0x3)             = 0 0
fsgetpath(0x16EFC70F8, 0x400, 0x16EFC70D8)             = 59 0
fsgetpath(0x16EFC7108, 0x400, 0x16EFC70E8)             = 14 0
csrctl(0x0, 0x16EFC750C, 0x4)          = -1 Err#1
__mac_syscall(0x181047D62, 0x2, 0x16EFC7450)           = 0 0
csrctl(0x0, 0x16EFC74FC, 0x4)          = -1 Err#1
__mac_syscall(0x181044B95, 0x5A, 0x16EFC7490)          = 0 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16EFC69F8, 0x16EFC69F0, 0x181046888, 0xD)               = 0 0
sysctl([CTL_KERN, 157, 0, 0, 0, 0] (2), 0x16EFC6AA8, 0x16EFC6AA0, 0x0, 0x0)           = 0 0
open("/\0", 0x20100000, 0x0)           = 3 0
openat(0x3, "System/Cryptexes/OS\0", 0x100000, 0x0)            = 4 0
dup(0x4, 0x0, 0x0)             = 5 0
fstatat64(0x4, 0x16EFC6581, 0x16EFC64F0)               = 0 0
openat(0x4, "System/Library/dyld/\0", 0x100000, 0x0)           = 6 0
fcntl(0x6, 0x32, 0x16EFC6580)          = 0 0
dup(0x6, 0x0, 0x0)             = 7 0
dup(0x5, 0x0, 0x0)             = 8 0
close(0x3)             = 0 0
close(0x5)             = 0 0
close(0x4)             = 0 0
close(0x6)             = 0 0
__mac_syscall(0x181047D62, 0x2, 0x16EFC6F70)           = 0 0
shared_region_check_np(0x16EFC6B90, 0x0, 0x0)          = 0 0
fsgetpath(0x16EFC7110, 0x400, 0x16EFC7038)             = 82 0
fcntl(0x8, 0x32, 0x16EFC7110)          = 0 0
close(0x8)             = 0 0
close(0x7)             = 0 0
getfsstat64(0x0, 0x0, 0x2)             = 10 0
getfsstat64(0x100E38050, 0x54B0, 0x2)          = 10 0
getattrlist("/\0", 0x16EFC7050, 0x16EFC6FC0)           = 0 0
stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/dyld_shared_cache_arm64e\0", 0x16EFC73B0, 0x0)
= 0 0
stat64("/Users/matveyd/CLionProjects/OS-labs-active/lab2/src/mutex\0", 0x16EFC6860, 0x0)               = 0 0
open("/Users/matveyd/CLionProjects/OS-labs-active/lab2/src/mutex\0", 0x0, 0x0)         = 3 0
mmap(0x0, 0x8908, 0x1, 0x40002, 0x3, 0x0)              = 0x100E38000 0
fcntl(0x3, 0x32, 0x16EFC6978)          = 0 0
```

close(0x3) = 0 0
munmap(0x100E38000, 0x8908) = 0 0
stat64("/Users/matveyd/CLionProjects/OS-labs-active/lab2/src/mutex\0", 0x16EFC6DD0, 0x0) = 0 0
open("@rpath/libclang_rt.tsan_osx_dynamic.dylib\0", 0x0, 0x0) = -1 Err#2
open("@rpath\0", 0x100000, 0x0) = -1 Err#2
stat64("/Users/matveyd/CLionProjects/OS-labs-active/lab2/src/libclang_rt.tsan_osx_dynamic.dylib\0", 0x16EFC5BF0, 0x0) = -1 Err#2
stat64("/Library/Developer/CommandLineTools/usr/lib/clang/16/lib/darwin/libclang_rt.tsan_osx_dynamic.dylib\0", 0x16EFC5BF0, 0x0) = 0 0
stat64("/Library/Developer/CommandLineTools/usr/lib/clang/16/lib/darwin/libclang_rt.tsan_osx_dynamic.dylib\0", 0x16EFC5620, 0x0) = 0 0
open("/Library/Developer/CommandLineTools/usr/lib/clang/16/lib/darwin/libclang_rt.tsan_osx_dynamic.dylib\0", 0x0, 0x0) = 3 0
mmap(0x0, 0x376070, 0x1, 0x40002, 0x3, 0x0) = 0x100E3C000 0
fcntl(0x3, 0x32, 0x16EFC5738) = 0 0
close(0x3) = 0 0
open("/Library/Developer/CommandLineTools/usr/lib/clang/16/lib/darwin/libclang_rt.tsan_osx_dynamic.dylib\0", 0x0, 0x0) = 3 0
fstat64(0x3, 0x16EFC4DC0, 0x0) = 0 0
fcntl(0x3, 0x61, 0x16EFC53B8) = 0 0
fcntl(0x3, 0x62, 0x16EFC53B8) = 0 0
mmap(0x1011B4000, 0x98000, 0x5, 0x40012, 0x3, 0x29C000) = 0x1011B4000 0
mmap(0x10124C000, 0x4000, 0x3, 0x40012, 0x3, 0x334000) = 0x10124C000 0
mmap(0x101250000, 0x8000, 0x3, 0x40012, 0x3, 0x338000) = 0x101250000 0
mmap(0x102694000, 0x38000, 0x1, 0x40012, 0x3, 0x340000) = 0x102694000 0
close(0x3) = 0 0
munmap(0x100E3C000, 0x376070) = 0 0
open("/Users/matveyd/CLionProjects/OS-labs-active/lab2/src/mutex\0", 0x0, 0x0) = 3 0
__mac_syscall(0x181047D62, 0x2, 0x16EFC41F0) = 0 0
map_with_linking_np(0x16EFC4050, 0x1, 0x16EFC4080) = 0 0
close(0x3) = 0 0
mprotect(0x100E2C000, 0x4000, 0x1) = 0 0
mprotect(0x10124C000, 0x4000, 0x1) = 0 0
open("/dev/dtracehelper\0", 0x2, 0x0) = 3 0
ioctl(0x3, 0x80086804, 0x16EFC3518) = 0 0
close(0x3) = 0 0
shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0) = 0 0
access("/AppleInternal/XBS/.isChrooted\0", 0x0, 0x0) = -1 Err#2
bsdthread_register(0x18134A0F4, 0x18134A0E8, 0x4000) = 1073746399 0
sysctl([CTL_HW, 7, 0, 0, 0, 0] (2), 0x1E646B2F0, 0x16EFC4068, 0x0, 0x0) = 0 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100E40000 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100E44000 0
stat64("/\0", 0x16EFC2360, 0x0) = 0 0
getattrlist("/Users\0", 0x18125ED20, 0x16EFC3C70) = 0 0
getattrlist("/Users/matveyd\0", 0x18125ED20, 0x16EFC3C70) = 0 0
getattrlist("/Users/matveyd/CLionProjects\0", 0x18125ED20, 0x16EFC3C70) = 0 0
getattrlist("/Users/matveyd/CLionProjects/OS-labs-active\0", 0x18125ED20, 0x16EFC3C70) = 0 0
getattrlist("/Users/matveyd/CLionProjects/OS-labs-active/lab2\0", 0x18125ED20, 0x16EFC3C70) = 0 0
getattrlist("/Users/matveyd/CLionProjects/OS-labs-active/lab2/src\0", 0x18125ED20, 0x16EFC3C70) = 0 0
getattrlist("/Users/matveyd/CLionProjects/OS-labs-active/lab2/src/mutex\0", 0x18125ED20, 0x16EFC3C70) = 0 0
munmap(0x100E44000, 0x4000) = 0 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100E44000 0
getattrlist("/Users\0", 0x18125ED20, 0x16EFC3C70) = 0 0
getattrlist("/Users/matveyd\0", 0x18125ED20, 0x16EFC3C70) = 0 0
getattrlist("/Users/matveyd/CLionProjects\0", 0x18125ED20, 0x16EFC3C70) = 0 0
getattrlist("/Users/matveyd/CLionProjects/OS-labs-active\0", 0x18125ED20, 0x16EFC3C70) = 0 0
getattrlist("/Users/matveyd/CLionProjects/OS-labs-active/lab2\0", 0x18125ED20, 0x16EFC3C70) = 0 0

getattrlist("/Users/matveyd/CLionProjects/OS-labs-active/lab2/src\0", 0x18125ED20, 0x16EFC3C70) = 0 0
getattrlist("/Users/matveyd/CLionProjects/OS-labs-active/lab2/src/mutex\0", 0x18125ED20, 0x16EFC3C70) = 0 0
munmap(0x100E44000, 0x4000) = 0 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100E44000 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16EFC4068, 0x16EFC4060, 0x10123F65E, 0xE) = 0 0
sysctl([CTL_KERN, 2, 0, 0, 0, 0] (2), 0x16EFC4100, 0x16EFC40F8, 0x0, 0x0) = 0 0
mmap(0x0, 0x800000, 0x0, 0x1042, 0x63000000, 0x0) = 0x1026CC000 0
mprotect(0x100E48000, 0x4000, 0x1) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100E4C000 0
munmap(0x100E4C000, 0xB4000) = 0 0
munmap(0x101000000, 0x4C000) = 0 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100E4C000 0
mmap(0x0, 0x390000, 0x3, 0x1002, 0x63000000, 0x0) = 0x102ECC000 0
getrlimit(0x1004, 0x16EFC4110, 0x0) = 0 0
setrlimit(0x1004, 0x16EFC4110, 0x0) = 0 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16EFC3E48, 0x16EFC3E40, 0x10123FA7B, 0x15) = 0 0
sysctl([CTL_KERN, 152, 0, 0, 0, 0] (2), 0x16EFC3EE0, 0x16EFC3F28, 0x0, 0x0) = 0 0
mmap(0x8000000000, 0x8000000000, 0x0, 0x1052, 0x63000000, 0x0) = 0x8000000000 0
mmap(0x100000000000, 0x200000000000, 0x0, 0x1052, 0x63000000, 0x0) = 0x100000000000 0
mmap(0x340000000000, 0x210000000000, 0x0, 0x1052, 0x63000000, 0x0) = 0x340000000000 0
mmap(0x568000000000, 0x248000000000, 0x0, 0x1052, 0x63000000, 0x0) = 0x568000000000 0
mmap(0x7C0000000000, 0x28000000000, 0x0, 0x1052, 0x63000000, 0x0) = 0x7C0000000000 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100E50000 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100E54000 0
mprotect(0x100E54000, 0x740, 0x1) = 0 0
mmap(0x10000000000, 0xF0000000000, 0x3, 0x1052, 0x63000000, 0x0) = 0x10000000000 0
mmap(0x300000000000, 0x40000000000, 0x3, 0x1052, 0x63000000, 0x0) = 0x300000000000 0
sigaltstack(0x0, 0x16EFC4120, 0x0) = 0 0
mmap(0x0, 0x80000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100E58000 0
sigaltstack(0x16EFC4138, 0x0, 0x0) = 0 0
sigaction(0xB, 0x16EFC40F8, 0x0) = 0 0
sigaction(0xA, 0x16EFC40F8, 0x0) = 0 0
sigaction(0x8, 0x16EFC40F8, 0x0) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x10325C000 0
munmap(0x10325C000, 0xA4000) = 0 0
munmap(0x103400000, 0x5C000) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x103400000 0
munmap(0x103500000, 0x100000) = 0 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100ED8000 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x103500000 0
munmap(0x103600000, 0x100000) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x103600000 0
munmap(0x103700000, 0x100000) = 0 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100EDC000 0
mmap(0x0, 0x160000, 0x3, 0x1002, 0x63000000, 0x0) = 0x101000000 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100EE0000 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x103700000 0
munmap(0x103800000, 0x100000) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x103800000 0
munmap(0x103900000, 0x100000) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x103900000 0
munmap(0x103A00000, 0x100000) = 0 0
munmap(0x100EE0000, 0x4000) = 0 0
munmap(0x100EDC000, 0x4000) = 0 0
getpid(0x0, 0x0, 0x0) = 4083 0
__mac_syscall(0x18D2F2505, 0x2, 0x16EFC3EE0) = 0 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100EDC000 0

__mac_syscall(0x18D2F2505, 0x2, 0x16EFC3EE0) = 0 0
__mac_syscall(0x18D2F2505, 0x2, 0x16EFC3EE0) = 0 0
__mac_syscall(0x18D2F2505, 0x2, 0x16EFC3EE0) = 0 0
__mac_syscall(0x18D2F2505, 0x2, 0x16EFC3EE0) = 0 0
__mac_syscall(0x18D2F2505, 0x2, 0x16EFC3EE0) = 0 0
stat64("/usr/bin/atos\0", 0x16EFC3FE0, 0x0) = 0 0
munmap(0x100EDC000, 0x4000) = 0 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100EDC000 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100EE0000 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x103A00000 0
munmap(0x103B00000, 0x100000) = 0 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100EE4000 0
mmap(0x0, 0x80000, 0x3, 0x1042, 0x63000000, 0x0) = 0x10325C000 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x103B00000 0
munmap(0x103C00000, 0x100000) = 0 0
getrlimit(0x1003, 0x16EFC4060, 0x0) = 0 0
mmap(0x0, 0x40000, 0x3, 0x1002, 0x63000000, 0x0) = 0x101160000 0
shm_open(0x1811E1F41, 0x0, 0x39FEED0) = 3 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x103C00000 0
munmap(0x103D00000, 0x100000) = 0 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100EE8000 0
mmap(0x0, 0x10000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100EEC000 0
mmap(0x0, 0x100000, 0x3, 0x1002, 0x63000000, 0x0) = 0x103D00000 0
mmap(0x0, 0x800000, 0x3, 0x1042, 0x63000000, 0x0) = 0x103E00000 0
fstat64(0x3, 0x16EFC3BE0, 0x0) = 0 0
mmap(0x0, 0x8000, 0x1, 0x40001, 0x3, 0x0) = 0x1011A0000 0
close(0x3) = 0 0
csops(0xFF3, 0x0, 0x16EFC3D1C) = 0 0
mmap(0x0, 0x18000, 0x3, 0x1002, 0x63000000, 0x0) = 0x1032DC000 0
ioctl(0x2, 0x4004667A, 0x16EFC3C8C) = 0 0
mprotect(0x104600000, 0x4000, 0x0) = 0 0
mprotect(0x10460C000, 0x4000, 0x0) = 0 0
mprotect(0x104610000, 0x4000, 0x0) = 0 0
mprotect(0x10461C000, 0x4000, 0x0) = 0 0
mprotect(0x104620000, 0x4000, 0x0) = 0 0
mprotect(0x10462C000, 0x4000, 0x0) = 0 0
mprotect(0x1011A8000, 0xC8, 0x1) = 0 0
mprotect(0x1011A8000, 0xC8, 0x3) = 0 0
mprotect(0x1011A8000, 0xC8, 0x1) = 0 0
mprotect(0x100E48000, 0x4000, 0x3) = 0 0
mprotect(0x100E48000, 0x4000, 0x1) = 0 0
mprotect(0x1032F4000, 0xC8, 0x1) = 0 0
madvise(0x802065E8000, 0x10000, 0x5) = 0 0
write(0x2, "mutex(4083,0x1e6463840) malloc: nano zone abandoned due to inability to reserve vm space.\n\0", 0x5A) = 90 0
proc_info(0x2, 0xFF3, 0x11) = 56 0
socket(0x1, 0x2, 0x0) = 3 0
fcntl(0x3, 0x2, 0x1) = 0 0
connect(0x3, 0x16EFC2C46, 0x6A) = 0 0
sendto(0x3, 0x16EFC2D30, 0xDE) = 222 0
issetugid(0x0, 0x0, 0x0) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x104630000 0
munmap(0x104630000, 0xD0000) = 0 0
munmap(0x104800000, 0x30000) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x104800000 0
munmap(0x104900000, 0x100000) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x104900000 0

munmap(0x104A00000, 0x100000) = 0 0
getentropy(0x16EFC3298, 0x20, 0x0) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x104A00000 0
munmap(0x104B00000, 0x100000) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x104B00000 0
munmap(0x104C00000, 0x100000) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x104C00000 0
munmap(0x104D00000, 0x100000) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x104D00000 0
munmap(0x104E00000, 0x100000) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x104E00000 0
munmap(0x104F00000, 0x100000) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x104F00000 0
munmap(0x105000000, 0x100000) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x105000000 0
munmap(0x105100000, 0x100000) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x105100000 0
munmap(0x105200000, 0x100000) = 0 0
getattrlist("/Users/matveyd/CLionProjects/OS-labs-active/lab2/src/mutex\0", 0x16EFC3B80, 0x16EFC3B9C) = 0 0
access("/Users/matveyd/CLionProjects/OS-labs-active/lab2/src\0", 0x4, 0x0) = 0 0
open("/Users/matveyd/CLionProjects/OS-labs-active/lab2/src\0", 0x0, 0x0) = 4 0
fstat64(0x4, 0x105103ED0, 0x0) = 0 0
csrctl(0x0, 0x16EFC3D6C, 0x4) = 0 0
fcntl(0x4, 0x32, 0x16EFC3A68) = 0 0
close(0x4) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x105200000 0
munmap(0x105300000, 0x100000) = 0 0
open("/Users/matveyd/CLionProjects/OS-labs-active/lab2/src/Info.plist\0", 0x0, 0x0) = -1 Err#2
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x105300000 0
munmap(0x105400000, 0x100000) = 0 0
proc_info(0x2, 0xFF3, 0xD) = 64 0
csops_audittoken(0xFF3, 0x10, 0x16EFC3DF0) = 0 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16EFC40E8, 0x16EFC40E0, 0x184A5BD3A, 0x15) = 0 0
sysctl([CTL_KERN, 155, 0, 0, 0, 0] (2), 0x16EFC41D8, 0x16EFC41D0, 0x0, 0x0) = 0 0
mmap(0x0, 0x200000, 0x3, 0x1002, 0x63000000, 0x0) = 0x105400000 0
munmap(0x105500000, 0x100000) = 0 0
__pthread_sigmask(0x3, 0x16EFD713C, 0x16EFD7138) = 0 0
bsdthread_create(0x1011E48CC, 0x16EFD7178, 0x16F177000) = 1862660096 0
__pthread_sigmask(0x3, 0x16EFD7138, 0x0) = 0 0
bsdthread_create(0x1011E48CC, 0x16EFD7178, 0x16F0EB000) = 1863233536 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x100EFC000 0
thread_selfid(0x0, 0x0, 0x0) = 83002 0
thread_selfid(0x0, 0x0, 0x0) = 83003 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x1011B0000 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x1032F4000 0
mmap(0x0, 0x80000, 0x3, 0x1042, 0x63000000, 0x0) = 0x104630000 0
mmap(0x0, 0x40000, 0x3, 0x1002, 0x63000000, 0x0) = 0x1046B0000 0
mmap(0x802DE0D4000, 0x100000, 0x3, 0x1052, 0x63000000, 0x0) = 0x802DE0D4000 0
mmap(0x0, 0x4000, 0x3, 0x1002, 0x63000000, 0x0) = 0x1032F8000 0
mmap(0x0, 0x80000, 0x3, 0x1042, 0x63000000, 0x0) = 0x105500000 0
mmap(0x0, 0x40000, 0x3, 0x1002, 0x63000000, 0x0) = 0x105580000 0
mmap(0x802DE1EC000, 0x100000, 0x3, 0x1052, 0x63000000, 0x0) = 0x802DE1EC000 0
psynch_mutexdrop(0x100E30000, 0x100, 0x100) = 0 0
psynch_mutexwait(0x100E30000, 0x102, 0x0) = 259 0
psynch_mutexdrop(0x100E30000, 0x200, 0x200) = 0 0
psynch_mutexwait(0x100E30000, 0x202, 0x100) = 515 0
psynch_mutexdrop(0x100E30000, 0x300, 0x300) = 0 0

psynch_mutexwait(0x100E30000, 0x302, 0x200) = 771 0
psynch_mutexdrop(0x100E30000, 0x400, 0x400) = 0 0
psynch_mutexwait(0x100E30000, 0x402, 0x300) = 1027 0
psynch_mutexdrop(0x100E30000, 0x500, 0x500) = 0 0
psynch_mutexwait(0x100E30000, 0x502, 0x400) = 1283 0
psynch_mutexdrop(0x100E30000, 0x600, 0x600) = 0 0
psynch_mutexwait(0x100E30000, 0x602, 0x500) = 1539 0
psynch_mutexdrop(0x100E30000, 0x700, 0x700) = 0 0
psynch_mutexwait(0x100E30000, 0x702, 0x600) = 1795 0
psynch_mutexdrop(0x100E30000, 0x800, 0x800) = 0 0
psynch_mutexwait(0x100E30000, 0x802, 0x700) = 2051 0
psynch_mutexdrop(0x100E30000, 0x900, 0x900) = 0 0
psynch_mutexwait(0x100E30000, 0x902, 0x800) = 2307 0
psynch_mutexdrop(0x100E30000, 0xA00, 0xA00) = 0 0
psynch_mutexwait(0x100E30000, 0xA02, 0x900) = 2563 0
psynch_mutexdrop(0x100E30000, 0xB00, 0xB00) = 0 0
psynch_mutexwait(0x100E30000, 0xB02, 0xA00) = 2819 0
psynch_mutexdrop(0x100E30000, 0xC00, 0xC00) = 0 0
psynch_mutexwait(0x100E30000, 0xC02, 0xB00) = 3075 0
__disable_threadsignal(0x1, 0x0, 0x0) = 0 0
mmap(0x802DE0D4000, 0x100000, 0x3, 0x1052, 0x63000000, 0x0) = 0x802DE0D4000 0
madvise(0x802DE0D0000, 0x104000, 0x5) = 0 0
munmap(0x104630000, 0x80000) = 0 0
munmap(0x1032F4000, 0x740) = 0 0
ulock_wake(0x1000002, 0x16F0EB034, 0x0) = 0 0
ulock_wait(0x1020002, 0x16F0EB034, 0x1E0B) = 0 0
__disable_threadsignal(0x1, 0x0, 0x0) = 0 0
mmap(0x802DE1EC000, 0x100000, 0x3, 0x1052, 0x63000000, 0x0) = 0x802DE1EC000 0
madvise(0x802DE1E8000, 0x104000, 0x5) = 0 0
munmap(0x105500000, 0x80000) = 0 0
munmap(0x1032F8000, 0x740) = 0 0
ulock_wake(0x1000002, 0x16F177034, 0x0) = 0 0
ulock_wait(0x1020002, 0x16F177034, 0xC03) = 0 0
write(0x1, "(-240.00 + -122.00i) \0", 0x15) = 21 0
write(0x1, "(70.00 + 141.00i) \0", 0x12) = 18 0
write(0x1, "(68.00 + 26.00i) \0", 0x11) = 17 0
write(0x1, "(-216.00 + -164.00i) \0", 0x15) = 21 0
write(0x1, "(484.00 + -57.00i) \0", 0x13) = 19 0
write(0x1, "(162.00 + -329.00i) \0", 0x14) = 20 0
write(0x1, "(-130.00 + -97.00i) \0", 0x14) = 20 0
write(0x1, "(107.00 + 22.00i) \0", 0x12) = 18 0
write(0x1, "(-451.00 + -34.00i) \0", 0x14) = 20 0
write(0x1, "(-241.00 + 408.00i) \0", 0x14) = 20 0
write(0x1, "(-266.00 + 0.00i) \0", 0x12) = 18 0
write(0x1, "(301.00 + 267.00i) \0", 0x13) = 19 0
write(0x1, "(226.00 + 1.00i) \0", 0x11) = 17 0
write(0x1, "(60.00 + -57.00i) \0", 0x12) = 18 0
write(0x1, "(120.00 + 31.00i) \0", 0x12) = 18 0
write(0x1, "(104.00 + 68.00i) \0", 0x12) = 18 0
write(0x1, "(252.00 + 166.00i) \0", 0x13) = 19 0
write(0x1, "(345.00 + -94.00i) \0", 0x13) = 19 0
write(0x1, "(-141.00 + 215.00i) \0", 0x14) = 20 0
write(0x1, "(-177.00 + 70.00i) \0", 0x13) = 19 0
write(0x1, "\n\0", 0x1) = 1 0
write(0x1, "(-60.00 + 329.00i) \0", 0x13) = 19 0
write(0x1, "(330.00 + -88.00i) \0", 0x13) = 19 0
write(0x1, "(-81.00 + -107.00i) \0", 0x14) = 20 0

```
write(0x1, "(-83.00 + 96.00i) \0", 0x12)              = 18 0
write(0x1, "(-319.00 + 327.00i) \0", 0x14)            = 20 0
write(0x1, "(-124.00 + 164.00i) \0", 0x14)            = 20 0
write(0x1, "(273.00 + 168.00i) \0", 0x13)             = 19 0
write(0x1, "(-106.00 + 362.00i) \0", 0x14)            = 20 0
write(0x1, "(-87.00 + 85.00i) \0", 0x12)              = 18 0
write(0x1, "(-267.00 + -278.00i) \0", 0x15)           = 21 0
write(0x1, "(265.00 + 119.00i) \0", 0x13)             = 19 0
write(0x1, "(-125.00 + 144.00i) \0", 0x14)            = 20 0
write(0x1, "(50.00 + 212.00i) \0", 0x12)              = 18 0
write(0x1, "(-96.00 + 137.00i) \0", 0x13)             = 19 0
write(0x1, "(-245.00 + 296.00i) \0", 0x14)            = 20 0
write(0x1, "(-172.00 + -211.00i) \0", 0x15)           = 21 0
write(0x1, "(31.00 + 32.00i) \0", 0x11)         = 17 0
write(0x1, "(-147.00 + 312.00i) \0", 0x14)            = 20 0
write(0x1, "(-103.00 + 58.00i) \0", 0x13)             = 19 0
write(0x1, "(-71.00 + -113.00i) \0", 0x14)            = 20 0
write(0x1, "\n\0", 0x1)         = 1 0
write(0x1, "(-220.00 + 45.00i) \0", 0x13)             = 19 0
write(0x1, "(-178.00 + -67.00i) \0", 0x14)            = 20 0
write(0x1, "(-262.00 + 252.00i) \0", 0x14)            = 20 0
…
Много write
…
write(0x1, "(-198.00 + -296.00i) \0", 0x15)           = 21 0
write(0x1, "(-65.00 + 28.00i) \0", 0x12)              = 18 0
write(0x1, "(-280.00 + -100.00i) \0", 0x15)           = 21 0
write(0x1, "(-221.00 + 189.00i) \0", 0x14)            = 20 0
write(0x1, "(-779.00 + 131.00i) \0", 0x14)            = 20 0
write(0x1, "(240.00 + -238.00i) \0", 0x14)            = 20 0
write(0x1, "(61.00 + 88.00i) \0", 0x11)         = 17 0
write(0x1, "(87.00 + -73.00i) \0", 0x12)              = 18 0
write(0x1, "(-206.00 + -588.00i) \0", 0x15)           = 21 0
write(0x1, "(-278.00 + -123.00i) \0", 0x15)           = 21 0
write(0x1, "(104.00 + -329.00i) \0", 0x14)            = 20 0
write(0x1, "(124.00 + 295.00i) \0", 0x13)             = 19 0
write(0x1, "(-55.00 + 236.00i) \0", 0x13)             = 19 0
write(0x1, "(-105.00 + -56.00i) \0", 0x14)            = 20 0
write(0x1, "(-238.00 + -140.00i) \0", 0x15)           = 21 0
write(0x1, "(104.00 + -233.00i) \0", 0x14)            = 20 0
write(0x1, "(248.00 + 153.00i) \0", 0x13)             = 19 0
write(0x1, "\n\0", 0x1)         = 1 0
write(0x1, "(-63.00 + -53.00i) \0", 0x13)             = 19 0
write(0x1, "(-17.00 + -176.00i) \0", 0x14)            = 20 0
write(0x1, "(14.00 + 174.00i) \0", 0x12)              = 18 0
write(0x1, "(-108.00 + -177.00i) \0", 0x15)           = 21 0
write(0x1, "(74.00 + -161.00i) \0", 0x13)             = 19 0
write(0x1, "(37.00 + -384.00i) \0", 0x13)             = 19 0
write(0x1, "(-283.00 + 0.00i) \0", 0x12)              = 18 0
write(0x1, "(171.00 + -195.00i) \0", 0x14)            = 20 0
write(0x1, "(-42.00 + 142.00i) \0", 0x13)             = 19 0
write(0x1, "(117.00 + -33.00i) \0", 0x13)             = 19 0
write(0x1, "(191.00 + -109.00i) \0", 0x14)            = 20 0
write(0x1, "(-137.00 + -44.00i) \0", 0x14)            = 20 0
write(0x1, "(9.00 + -25.00i) \0", 0x11)         = 17 0
write(0x1, "(134.00 + -282.00i) \0", 0x14)            = 20 0
write(0x1, "(3.00 + 207.00i) \0", 0x11)         = 17 0
```

```
write(0x1, "(45.00 + -43.00i) \0", 0x12)              = 18 0
write(0x1, "(-189.00 + -61.00i) \0", 0x14)            = 20 0
write(0x1, "(-68.00 + 183.00i) \0", 0x13)             = 19 0
write(0x1, "(-39.00 + -170.00i) \0", 0x14)            = 20 0
write(0x1, "(115.00 + 223.00i) \0", 0x13)             = 19 0
write(0x1, "\n\0", 0x1)          = 1 0
```

Вывод

В процессе выполнения данной лабораторной работы я освоил навыки разработки программ, использующих многопоточность, а также научился эффективно синхронизировать потоки между собой. Анализ результатов тестирования программы позволил изучить влияние количества потоков на её производительность и коэффициент ускорения. Было установлено, что при большом объёме данных значительное число потоков может существенно ускорить выполнение задачи, однако эффективность использования ресурсов возрастает только при количестве потоков, сопоставимом с числом логических ядер процессора. Работа над лабораторной стала ценным опытом, так как она впервые позволила мне погрузиться в изучение многопоточности и механизмов синхронизации в языке программирования Си.