Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №2 по курсу

«Операционные системы»

Группа: М8О-211Б-23

Студент: Савков И.И.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 01.12.24

Москва, 2024

Постановка задачи

Цель работы:

Целью является приобретение практических навыков в:

- **Управление потоками в ОС**
- **Обеспечение синхронизации между потоками**

Задание:

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

Вариант 20) Дан массив координат (x, y, z). Необходимо найти три точки, которые образуют треугольник максимальной площади

Общий метод и алгоритм решения

Использованные системные вызовы:

- void exit(int __status); – выполняет немедленное завершение программы. Все используемые программой потоки закрываются, и временные файлы удаляются, управление возвращается ОС или другой программе.
- int pthread_create(pthread_t *__restrict__newthread, const pthread_attr_t *__restrict_____attr, void *(*__start_routine)(void *), void *__restrict__arg) — создаёт поток с рутиной (стартовой функцией) и заданными аргументами
- int pthread_join(pthread_t __th, void **_thread_return) — дожидается завершения потока.
- pthread_exit() - Функция pthread_exit() завершает вызывающий поток и возвращает значение через retval, которое (если поток можно объединить) доступно другому потоку в том же процессе, который вызывает pthread_join(3).

Программа получает на вход два аргумента – имя файла с координатами и максимальное количество потоков. Далее создаётся и заполняется массив для хранения всех точек.

После создаётся нужное количество потоков, которые выполняют функция потока find_max_area_thread

Функция принимает структуру **ThreadData**, которая содержит:

- **massive** — указатель на массив точек.
- **num_threads** — общее количество потоков.
- **thread_id** — уникальный идентификатор потока (от 0 до num_threads - 1).
- **max_area** — переменная для хранения максимальной площади треугольника, найденного этим потоком.
- **max_points[3]** — массив указателей на три точки, образующие треугольник с максимальной площадью.

1. **Циклическое распределение задач**
   - Комбинации точек (i, j, k) распределяются между потоками с помощью формулы:

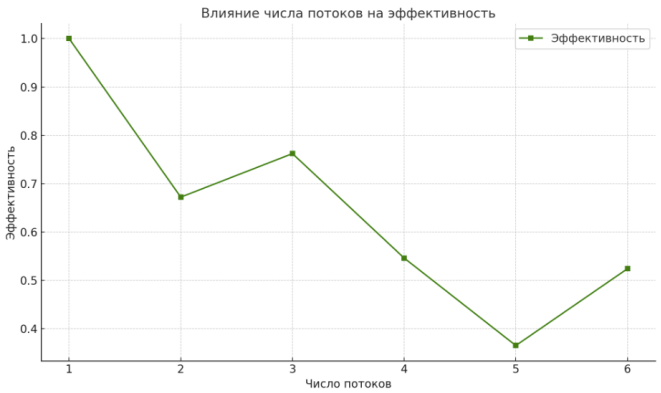
     (i + j + k) % num_threads == thread_id

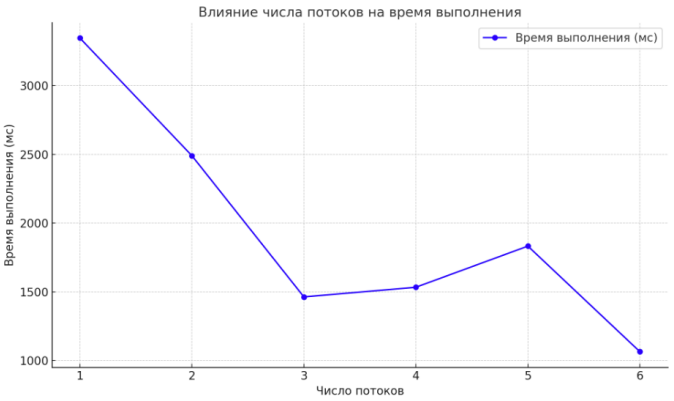
   - Если комбинация не соответствует текущему потоку (thread_id), она пропускается (continue).
   - Это упрощает распределение работы без необходимости разделять массив вручную.
2. **Параллельная обработка**
   - Каждый поток работает с разными комбинациями точек, что ускоряет поиск максимальной площади.
3. **Локальное хранение результатов**
   - Каждый поток сохраняет свою максимальную площадь и точки в структуру ThreadData.
4. **Избежание конфликтов доступа к памяти**
   - Потоки не делят общий ресурс, так как работают с независимыми частями данных.

После завершения всех потоков главный поток:

1. Ждёт окончания каждого потока с помощью `pthread_join`.
2. Сравнивает максимальные площади, найденные каждым потоком, чтобы выбрать глобальный максимум.

Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	3346,24	1	1
2	2490,55	1,34	0,672
3	1463,02	2,29	0,762
4	1533,44	2,18	0,546
5	1832,66	1,83	0,365
6	1064,49	3,14	0,524





Код программы

functions.h:

```c
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <math.h>
#include <pthread.h>


typedef enum {
    OK,
    INVALID_ARGS,
    MEMORY_ERROR,
    INVALID_FILE,
    UNCORECT_TYPE_OF_NUMS,
    NULL_PTR,
} state;


typedef struct {
    double x, y, z;
} Point;

typedef struct {
    size_t size;
    size_t capacity;
    Point **array;
} PointMassive;

// === Потоки ====
typedef struct {
    // Массив точек
    PointMassive *massive;
    int num_threads; // Общее количество потоков
    int thread_id; // Уникальный ID потока
    double max_area; // Максимальная площадь, найденная этим потоком
    Point *max_points[3]; // Точки, образующие треугольник с максимальной площадью
} ThreadData;

// === Потоки ====

state is_digital(const char *str, double *x);

state get_word(FILE *input, char **buffer, char *c);

state get_point(FILE *file, double *x, double *y, double *z, char *symbol);
```

```c
state add_points(FILE *file, PointMassive **massive);

state insert(PointMassive **massive, double x, double y, double z);

void freePointMassive(PointMassive *massive);

PointMassive *create_PointMassive(int capacity);

double get_area(double x, double y, double z);

double vector_length(double x, double y, double z);

double triangle_area(Point *a, Point *b, Point *c);

void *find_max_area_thread(void *arg);


state is_digital(const char *str, double *x) {
    char *endptr;
    double temp = strtod(str, &endptr);

    if (*endptr != '\0') {
        return UNCORECT_TYPE_OF_NUMS;
    }

    *x = temp;
    return OK;
}

state get_word(FILE *input, char **buffer, char *c) {
    int capacity = 2;

    if (input == NULL || *buffer == NULL)
        return NULL_PTR;

    int index = 0;
    *c = (char) fgetc(input);

    while (*c != ' ' && *c != '\n' && *c != '\t' && *c != EOF) {
        if (capacity == index) {
            capacity *= 2;
            char *temp_buffer = (char *) realloc(*buffer, capacity);
            if (temp_buffer == NULL) {
                free(*buffer);
                return MEMORY_ERROR;
            }
            *buffer = temp_buffer;
        }

        (*buffer)[index++] = *c;
        *c = (char) fgetc(input);
    }

    (*buffer)[index] = '\0';
    return OK;
}

state get_point(FILE *file, double *x, double *y, double *z, char *symbol) {
    char *str_x, *str_y, *str_z;

    str_x = (char *) malloc(sizeof(char) * 128);
    str_y = (char *) malloc(sizeof(char) * 128);
    str_z = (char *) malloc(sizeof(char) * 128);

    if (get_word(file, &str_x, symbol) == MEMORY_ERROR ||
        get_word(file, &str_y, symbol) == MEMORY_ERROR ||
        get_word(file, &str_z, symbol) == MEMORY_ERROR) {
        free(str_x);
        free(str_y);
        free(str_z);
```

```
        return MEMORY_ERROR;
    }

    if (is_digital(str_x, x) == UNCORECT_TYPE_OF_NUMS ||
        is_digital(str_y, y) == UNCORECT_TYPE_OF_NUMS ||
        is_digital(str_z, z) == UNCORECT_TYPE_OF_NUMS) {
        free(str_x);
        free(str_y);
        free(str_z);

        return UNCORECT_TYPE_OF_NUMS;
    }

    free(str_x);
    free(str_y);
    free(str_z);
    return OK;
}

PointMassive *create_PointMassive(int capacity) {
    PointMassive *massive = (PointMassive *) malloc(sizeof(PointMassive));
    if (massive == NULL) {
        return NULL;
    }

    massive->size = 0;
    massive->capacity = capacity;
    massive->array = (Point **) malloc(sizeof(Point *) * capacity);

    if (massive->array == NULL) {
        free(massive);
        return NULL;
    }

    for (int i = 0; i < capacity; i++) {
        massive->array[i] = NULL;
    }

    return massive;
}

state insert(PointMassive **massive, double x, double y, double z) {
    Point *element = (Point *) malloc(sizeof(Point));

    PointMassive *masive_p = (*massive);
    if (element == NULL) {
        return MEMORY_ERROR;
    }

    if (masive_p->size == masive_p->capacity) {
        masive_p->capacity *= 2;
        Point **temp_buffer = (Point **) realloc(masive_p->array, masive_p->capacity *
sizeof(Point *));
        if (temp_buffer == NULL) {
            free(element);
            return MEMORY_ERROR;
        }
        masive_p->array = temp_buffer;
    }

    element->x = x;
    element->y = y;
    element->z = z;

    const size_t index = masive_p->size;
    masive_p->array[index] = element;

    masive_p->size++;
    return OK;
}
```

```c
void freePointMassive(PointMassive *massive) {
    if (massive == NULL) return;

    for (int i = 0; i < massive->size; i++) {
        free(massive->array[i]);
    }
    free(massive->array);
    free(massive);
}


state add_points(FILE *file, PointMassive **massive) {
    while (!feof(file)) {
        double x, y, z;
        char symbol;

        state result = get_point(file, &x, &y, &z, &symbol);
        if (result == MEMORY_ERROR) {
            freePointMassive(*massive);
            return MEMORY_ERROR;
        }
        if (result == UNCORECT_TYPE_OF_NUMS) {
            freePointMassive(*massive);
            return UNCORECT_TYPE_OF_NUMS;
        }

        result = insert(massive, x, y, z);
        if (result == MEMORY_ERROR) {
            freePointMassive(*massive);
            return MEMORY_ERROR;
        }
    }

    return OK;
}

// Вычисление длины вектора
double vector_length(double x, double y, double z) {
    return sqrt(x * x + y * y + z * z);
}

// Вычисление площади треугольника, заданного тремя точками
double triangle_area(Point *a, Point *b, Point *c) {
    // Векторы AB и AC
    double ab_x = b->x - a->x, ab_y = b->y - a->y, ab_z = b->z - a->z;
    double ac_x = c->x - a->x, ac_y = c->y - a->y, ac_z = c->z - a->z;

    // Векторное произведение AB × AC
    double cross_x = ab_y * ac_z - ab_z * ac_y;
    double cross_y = ab_z * ac_x - ab_x * ac_z;
    double cross_z = ab_x * ac_y - ab_y * ac_x;

    // Длина вектора (модуль векторного произведения)
    double cross_length = vector_length(cross_x, cross_y, cross_z);

    // Площадь треугольника
    return 0.5 * cross_length;
}


// === Потоки ====
void *find_max_area_thread(void *arg) {
    ThreadData *data = (ThreadData *) arg;
    PointMassive *massive = data->massive;
    int num_points = massive->size;

    double max_area = 0.0;
    Point *max_points[3] = {NULL, NULL, NULL};
```

```
    // Циклическое распределение работы
    for (int i = 0; i < num_points - 2; i++) {
        for (int j = i + 1; j < num_points - 1; j++) {
            for (int k = j + 1; k < num_points; k++) {
                if ((i + j + k) % data->num_threads != data->thread_id) {
                    continue; // Эта комбинация не для данного потока
                }
                double area = triangle_area(massive->array[i], massive->array[j], massive-
>array[k]);
                if (area > max_area) {
                    max_area = area;
                    max_points[0] = massive->array[i];
                    max_points[1] = massive->array[j];
                    max_points[2] = massive->array[k];
                }
            }
        }
    }

    data->max_area = max_area;
    memcpy(data->max_points, max_points, sizeof(max_points));
    pthread_exit(NULL);
}
// === Потоки ====

#endif //FUNCTIONS_H
```

parent:

```
# #include "functions.h"
#include "sys/time.h"

int main(const int argc, char *argv[]) {
    char *input_path;
    PointMassive *massive;
    state result;

    //Засекаем время выполнения
    struct timeval start, end;
    gettimeofday(&start, NULL);



    if (argc != 3) {
        printf("ERROR: INVALID_ARGS.\n");
        exit(INVALID_ARGS);
    }

    input_path = argv[1];
    FILE *file = fopen(input_path, "r");
    if (file == NULL) {
        printf("ERROR: INVALID_FILE.\n");
        exit(INVALID_FILE);
    }

    massive = create_PointMassive(128);

    result = add_points(file, &massive);
    if (result == MEMORY_ERROR) {
        fclose(file);
        printf("ERROR: MEMORY_ERROR.\n");
        exit(MEMORY_ERROR);
    }
    if (result == UNCORECT_TYPE_OF_NUMS) {
        fclose(file);
        printf("ERROR: UNCORECT_TYPE_OF_NUMS.\n");
        exit(UNCORECT_TYPE_OF_NUMS);
    }
    fclose(file);
```

```c
    // === Потоки ====
    double arv_threads;
    is_digital(argv[2], &arv_threads);


    int num_threads = (int) arv_threads; // Число потоков

    pthread_t threads[num_threads];
    ThreadData thread_data[num_threads];

    for (int t = 0; t < num_threads; t++) {
        thread_data[t].massive = massive;
        thread_data[t].num_threads = num_threads;
        thread_data[t].thread_id = t;

        // printf("Thread %d started\n", t + 1);
        pthread_create(&threads[t], NULL, find_max_area_thread, &thread_data[t]);
    }

    // Объединение результатов
    double max_area = 0.0;
    Point *max_points[3] = {NULL, NULL, NULL};
    for (int t = 0; t < num_threads; t++) {
        pthread_join(threads[t], NULL);
        // printf("Thread %d finished\n", t + 1);
        if (thread_data[t].max_area > max_area) {
            max_area = thread_data[t].max_area;
            memcpy(max_points, thread_data[t].max_points, sizeof(max_points));
        }
    }
    // === Потоки ====


    // Окончание подсчёта времени выполнения
    gettimeofday(&end, NULL);
    double delta = ((end.tv_sec  - start.tv_sec) * 1000000u +
       end.tv_usec - start.tv_usec) / 1.e6;

    // Вывод результата
    printf("Max area: %.2f\n", max_area);
    printf("Points: (%.2f, %.2f, %.2f), (%.2f, %.2f, %.2f), (%.2f, %.2f, %.2f)\n",
           max_points[0]->x, max_points[0]->y, max_points[0]->z,
           max_points[1]->x, max_points[1]->y, max_points[1]->z,
           max_points[2]->x, max_points[2]->y, max_points[2]->z);

    freePointMassive(massive);


    printf("Time take: %0.6f\n seconds", delta);

    return 0;
}
```

Протокол работы программы

```
goldglaid@GoldGlaid:~/OSLabs/lab2$ ./parent file.txt 1
Max area: 29923.84
Points: (87.00, 98.00, 99.00), (87.00, -98.00, -99.00), (-99.00, 74.00, -78.00)
Time take: 3.346240
 secondsgoldglaid@GoldGlaid:~/OSLabs/lab2$ ./parent file.txt 2
Max area: 29923.84
Points: (87.00, 98.00, 99.00), (87.00, -98.00, -99.00), (-99.00, 74.00, -78.00)
Time take: 2.490555
 secondsgoldglaid@GoldGlaid:~/OSLabs/lab2$ ./parent file.txt 3
Max area: 29923.84
Points: (87.00, 98.00, 99.00), (87.00, -98.00, -99.00), (-99.00, 74.00, -78.00)
Time take: 1.463022
 secondsgoldglaid@GoldGlaid:~/OSLabs/lab2$ ./parent file.txt 4
Max area: 29923.84
Points: (87.00, 98.00, 99.00), (87.00, -98.00, -99.00), (-99.00, 74.00, -78.00)
Time take: 1.533441
 secondsgoldglaid@GoldGlaid:~/OSLabs/lab2$ ./parent file.txt 5
Max area: 29923.84
Points: (87.00, 98.00, 99.00), (87.00, -98.00, -99.00), (-99.00, 74.00, -78.00)
Time take: 1.832657
 secondsgoldglaid@GoldGlaid:~/OSLabs/lab2$ ./parent file.txt 6
Max area: 29923.84
Points: (87.00, 98.00, 99.00), (87.00, -98.00, -99.00), (-99.00, 74.00, -78.00)
Time take: 1.064495
goldglaid@GoldGlaid:~/OSLabs/lab2$ strace ./parent file.txt 6

execve("./parent", ["./parent", "file.txt", "6"], 0x7fffd8c08030 /* 26 vars */) = 0
brk(NULL)                               = 0x55c1c41c5000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fffc5aac800) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f0449985000
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=18567, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 18567, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f0449980000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.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\0\0\0\0\0\0\0"..., 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=940560, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 942344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f0449899000
mmap(0x7f04498a7000, 507904, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe000) = 0x7f04498a7000
mmap(0x7f0449923000, 372736, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x8a000) = 0x7f0449923000
mmap(0x7f044997e000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe4000) = 0x7f044997e000
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\0P\237\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\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\0"..., 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) =
68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
```

```
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f0449670000
mprotect(0x7f0449698000, 2023424, PROT_NONE) = 0
mmap(0x7f0449698000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f0449698000
mmap(0x7f044982d000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7f044982d000
mmap(0x7f0449886000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f0449886000
mmap(0x7f044988c000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f044988c000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f044966d000
arch_prctl(ARCH_SET_FS, 0x7f044966d740) = 0
set_tid_address(0x7f044966da10)         = 70802
set_robust_list(0x7f044966da20, 24)     = 0
rseq(0x7f044966e0e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f0449886000, 16384, PROT_READ) = 0
mprotect(0x7f044997e000, 4096, PROT_READ) = 0
mprotect(0x55c19c4e5000, 4096, PROT_READ) = 0
mprotect(0x7f04499bf000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f0449980000, 18567)           = 0
getrandom("\xfa\x7d\xbf\x65\x0d\xc8\x2d\x47", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x55c1c41c5000
brk(0x55c1c41e6000)                     = 0x55c1c41e6000
openat(AT_FDCWD, "file.txt", O_RDONLY)  = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=10149, ...}, AT_EMPTY_PATH) = 0
read(3, "-41 -88 82\n93 42 -45\n-24 66 94\n3"..., 4096) = 4096
read(3, " 94\n89 -70 34\n16 -94 -86\n71 15 5"..., 4096) = 4096
read(3, "7 13 -71\n88 -2 63\n0 -66 -12\n62 -"..., 4096) = 1957
read(3, "", 4096)                       = 0
close(3)                                = 0
rt_sigaction(SIGRT_1, {sa_handler=0x7f0449701870, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f04496b2520},
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) =
0x7f0448e6c000
mprotect(0x7f0448e6d000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|C
LONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f044966c910, parent_tid=0x7f044966c910, exit_signal=0, stack=0x7f0448e6c000,
stack_size=0x7fff00, tls=0x7f044966c640} => {parent_tid=[70803]}, 88) = 70803
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f044866b000
mprotect(0x7f044866c000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|C
LONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f0448e6b910, parent_tid=0x7f0448e6b910, exit_signal=0, stack=0x7f044866b000,
stack_size=0x7fff00, tls=0x7f0448e6b640} => {parent_tid=[70804]}, 88) = 70804
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
```

0x7f0447e6a000
mprotect(0x7f0447e6b000, 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=0x7f044866a910, parent_tid=0x7f044866a910, exit_signal=0, stack=0x7f0447e6a000, stack_size=0x7fff00, tls=0x7f044866a640} => {parent_tid=[70805]}, 88) = 70805
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f0447669000
mprotect(0x7f044766a000, 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=0x7f0447e69910, parent_tid=0x7f0447e69910, exit_signal=0, stack=0x7f0447669000, stack_size=0x7fff00, tls=0x7f0447e69640} => {parent_tid=[70806]}, 88) = 70806
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f0446e68000
mprotect(0x7f0446e69000, 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=0x7f0447668910, parent_tid=0x7f0447668910, exit_signal=0, stack=0x7f0446e68000, stack_size=0x7fff00, tls=0x7f0447668640} => {parent_tid=[70807]}, 88) = 70807
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f0446667000
mprotect(0x7f0446668000, 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=0x7f0446e67910, parent_tid=0x7f0446e67910, exit_signal=0, stack=0x7f0446667000, stack_size=0x7fff00, tls=0x7f0446e67640} => {parent_tid=[70808]}, 88) = 70808
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
futex(0x7f044966c910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 70803, NULL, FUTEX_BITSET_MATCH_ANY) = 0
futex(0x7f0448e6b910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 70804, NULL, FUTEX_BITSET_MATCH_ANY) = 0
munmap(0x7f0448e6c000, 8392704)   = 0
futex(0x7f0446e67910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 70808, NULL, FUTEX_BITSET_MATCH_ANY) = 0
munmap(0x7f044866b000, 8392704)   = 0
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x3), ...}, AT_EMPTY_PATH) = 0
write(1, "Max area: 29923.84\n", 19Max area: 29923.84
)   = 19
write(1, "Points: (87.00, 98.00, 99.00), ("..., 80Points: (87.00, 98.00, 99.00), (87.00, -98.00, -99.00), (-99.00, 74.00, -78.00)
) = 80
write(1, "Time take: 1.234773\n", 20Time take: 1.234773
)   = 20
write(1, " seconds", 8 seconds)   = 8
exit_group(0)   = ?
+++ exited with 0 +++
goldglaid@GoldGlaid:~/OSLabs/lab2$

Вывод

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