

**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»**

Выполнил: А. В. Маркелов
Группа: М8О-207БВ-24
Преподаватель: Е. С. Миронов

Москва, 2025

Условие

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

Вариант: 20

Метод решения

При запуске программы через командную строку подаются три значения: выполняемая программа, количество потоков, путь к файлу с входными данными. Программа считывает в массив координаты точек из указанного при запуске файла, подсчитывает общее количество комбинаций точек образующих треугольники. Далее программа распределяет комбинации по потокам, инициализирует потоки и запускает их, передавая входные данные. Потоки подсчитывают площади треугольников с вершинами в точках из переданных комбинаций и, защищая мьютексами, обновляют глобальную переменную содержащую максимальную площадь. При успешном выполнении программа выводит результаты: максимальная площадь треугольника и координаты вершин этого треугольника

Описание программы

Входные данные: координаты точек (x, y, z) записанные в текстовый файл.

Задачи обработки аргументов запуска программы, распределения входных данных по потокам и управление потоками реализованы в файле task.c В файлах thread.h, mutex.h объявлены структуры данных и функции-обёртки для кроссплатформенного взаимодействия с потоками и мьютексами. В файле triangle_finder.h объявлены структуры данных необходимые для хранения координат точек и функции чтения и сохранения в массив вводных данных из файла, подсчета площадей треугольников из прочитанных точек.

В thread_linux.c, mutex_linux.c описана реализация функций для работы с потоками и мьютексами на POSIX-системах. В triangle_finder_linux.c реализованы для POSIX-систем структуры данных и функции объявленные в triangle_finder.h.

Результаты

После запуска программы происходит чтение файла с вводными данными и запись полученных координат точек в массив points. Далее подсчитывается количество всевозможных комбинаций точек иalloцируется память под потоки и их аргументы. Множество комбинаций разбивается на равные диапазоны и распределяются по потокам. Потоки инициализируются, принимают аргументы и выполняют подсчет площадей по комбинациям. Защищая мьютексом, сравнивают полученную максимальную площадь треугольника с вершинами в точках выделенного под поток диапазона точек с глобальной переменной мак-

симальной площади, и если локальное значение больше, перезаписывают эту глобальную переменную. После завершения работы всех созданных потоков выводятся результаты работы программы.

Пример работы:

Ввод:

./task 4 test_points.txt

(Содержимое test_points.txt:

0.0 0.0 0.0

1.0 0.0 0.0

0.0 1.0 0.0

0.0 0.0 1.0

2.0 2.0 0.0

3.0 1.0 2.0

1.5 3.5 1.0

4.0 2.5 3.0

-1.0 -1.0 0.0

2.5 0.5 4.0)

Вывод:

Загружено 10 точек

Всего комбинаций: 120

Поток 0: комбинации 0 - 30 (всего 30)

Поток 1: комбинации 30 - 60 (всего 30)

Поток 1: новый максимум 8.038968

Поток 0 завершен (обработано 30)

Поток 2 завершен (обработано 30)

Поток 1 завершен (обработано 30)

Поток 2: комбинации 60 - 90 (всего 30)

Поток 3: комбинации 90 - 120 (всего 30)

Поток 3: новый максимум 10.706307

Поток 3 завершен (обработано 30)

РЕЗУЛЬТАТЫ

Максимальная площадь: 10.706307

Вершины:

P1: (1.500000, 3.500000, 1.000000)

P2: (-1.000000, -1.000000, 0.000000)

P3: (2.500000, 0.500000, 4.000000)

В результате проведенного исследования (запуск программы при 25 входных точках и использовании от 1 до 16 потоков) выяснена зависимость ускорения и эффективности от входных данных и количества потоков. При многопоточности происходит рост ускорения работы программы, при 8 потоках наблюдается максимальное ускорение работы программы (2.2x) относительно скорости работы программы при однопоточном запуске.

При большем количестве задействованных потоков (более 8) видна тенденция снижения ускорения работы из-за различных издержек (синхронизация через мьютексы, создание и управление потоками, архитектурные ограничения). Эффективность падает так как при увеличении количества потоков при неизменном количестве обрабатываемых данных расходы (синхронизация, управление потоками, конкуренция за кэш и память) растут быстрее, чем выигрыши от распараллеливания.

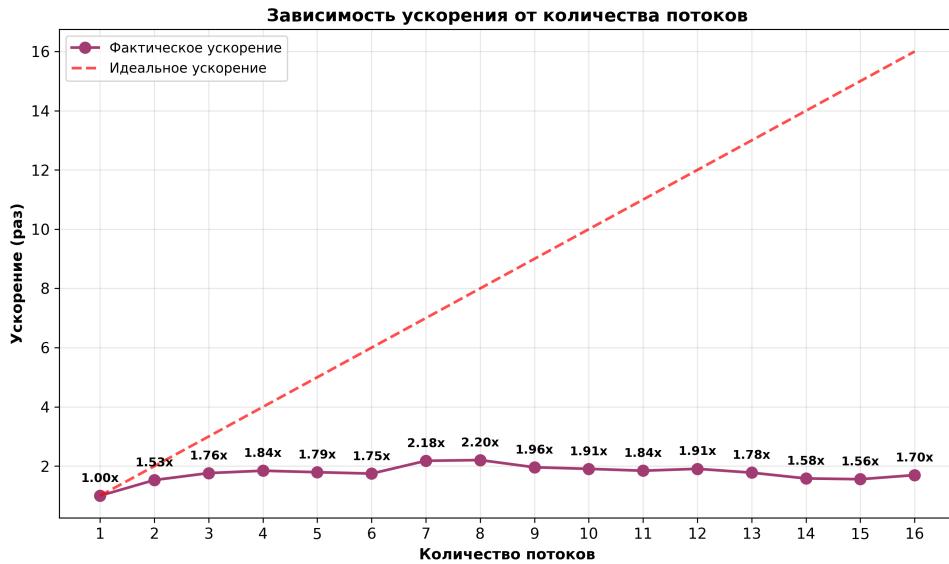


Рис. 1: График ускорения

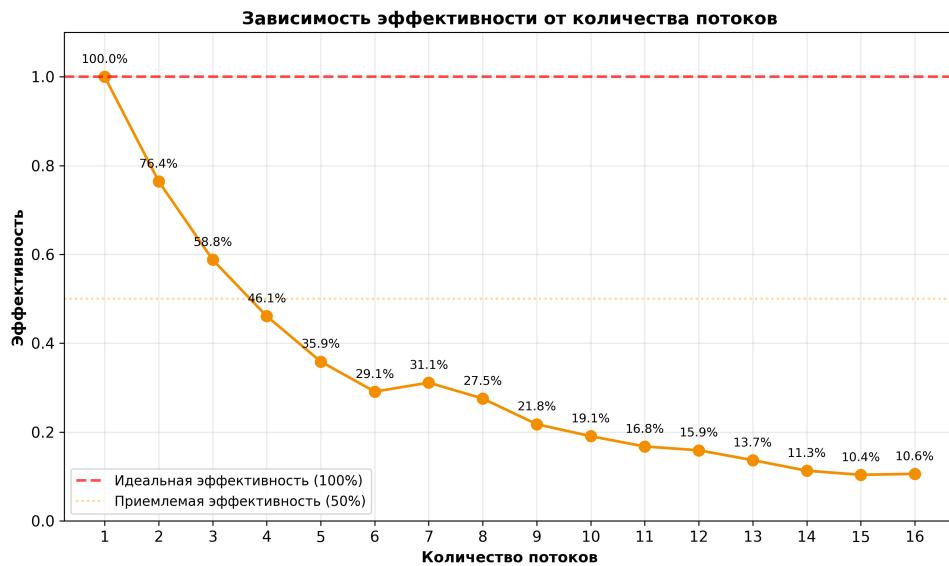


Рис. 2: График эффективности

Выводы

В рамках выполнения лабораторной работы реализованы:

- Многопоточный перебор сочетаний через POSIX-потоки (pthread) с распределением диапазонов между потоками.

- Защищенный мьютексом (`pthread_mutex_t`) глобальный максимум площади, что обеспечивает корректность при одновременном обновлении из разных потоков.

- Проведено исследование зависимости ускорения и эффективности от входных данных и количества потоков. Для 25 точек найдено максимальное ускорение (2.2x) относительно однопоточного запуска, достигаемое при 8 потоках. Изучена эффективность и найдены причины её снижения с увеличением количества потоков.

Поставленные цели достигнуты, лабораторная работа успешно выполнена. В процессе её выполнения я научился:

- Правильно распараллеливать задачу полного перебора на независимые потоки с распределением диапазонов.

- Использовать POSIX-потоки (`pthread_create`, `pthread_join`) и мьютексы (`pthread_mutex_*`) для синхронизации.

- Создавать кроссплатформенную программу, имеющую возможность запуска без изменений header-файлов (`thread.h`, `mutex.h`, `triangle_finder.h`) и главного файла задания (`task.c`) как на POSIX-системах так и на не POSIX-системах.

- Анализировать производительность: вычислять ускорение, эффективность.

- Интегрировать результаты в отчёт: код программы, графики и подробный анализ, обосновывающий наблюдаемое поведение алгоритма.

В итоге лабораторная работа позволила не только закрепить навыки работы с потоками и синхронизацией, но и приобрести опыт в проведении исследования производительности многопоточных приложений.

Исходная программа

```
#include "thread.h"
#include "triangle_finder.h"

#define MAX_THREADS 64

Triangle best_triangle;
best_triangle.area = -1.0;
pthread_mutex_t mutex;

void get_combination_by_index(int num_points,int combo_i,int*
i,int* j,
                                int* k) {
    int count = 0;
    for (int ii = 0; ii <num_points; ii++) {
        for (int jj = ii + 1; jj <num_points; jj++) {
            for (int kk = jj + 1; kk <num_points; kk++) {
                if (count == combo_i) {
                    *i = ii;
                    *j = jj;
                    *k = kk;
                    return;
                }
                count++;
            }
        }
    }
}

void* worker_thread(void* arg) {
    ThreadArgs* args = (ThreadArgs*)arg;
    Triangle local_best;
    local_best.area = -1.0;

    int n = args->num_points;
    Point3D* points = args->points;
    long long count = 0;

    for (int combo_i = args->start_i; combo_i <args->end_i;
combo_i++) {
        int i;
        int j;
        int k;
        get_combination_by_index(args->num_points,combo_i,&i,&j,&k);
        double area =
CalculateTriangleArea(points[i],points[j],points[k]);

        if (area >local_best.area) {
```

```

        local_best.area = area;
        local_best.vertices[0] = points[i];
        local_best.vertices[1] = points[j];
        local_best.vertices[2] = points[k];
    }
    count++;
}

LockMutex(&mutex);
if (local_best.area >best_triangle.area) {
    best_triangle = local_best;
    printf("Поток %d: новый максимум
%.6f\n",args->thread_id,local_best.area);
}
UnlockMutex(&mutex);

printf("Поток %d завершен (обработано
%lld)\n",args->thread_id,count);
return NULL;
}

int main(int argc,char* argv[]) {
    Point3D* points;
    int num_points;

    if (argc == 3 && strcmp(argv[2],"-") != 0) {
        ReadPointsFromFile(argv[2],&points,&num_points);
    } else {
        printf("usage: %s <threads>[file|num_points]\n",argv[0]);
        return 1;
    }

    int num_threads = atoi(argv[1]);
    if (num_threads <= 0 || num_threads >MAX_THREADS) {
        num_threads = MAX_THREADS;
    }

    long long total_combinations =
        (long long)num_points * (num_points -1) * (num_points -2) /
6;
    printf("Всего комбинаций: %lld\n\n",total_combinations);

    if (CreateMutex(&mutex) != 0) {
        perror("error while CreateMutex");
        free(points);
        return EXIT_FAILURE;
    }
}

```

```

Thread* threads = malloc(num_threads * sizeof(Thread));
ThreadArgs* args = malloc(num_threads * sizeof(ThreadArgs));
if (!threads || !args) {
    perror("error while malloc threads/args");
    free(points);
    DestroyMutex(&mutex);
    return EXIT_FAILURE;
}

int base_combo = total_combinations / num_threads;
int extra_combo = total_combinations % num_threads;
int start = 0;
for (int i = 0; i < num_threads; i++) {
    args[i].thread_id = i;
    args[i].points = points;
    args[i].num_points = num_points;

    int count;
    if (i < extra_combo) {
        count = base_combo + 1;
    } else {
        count = base_combo;
    }
    args[i].start_i = start;
    args[i].end_i = start + count;
    start += count;

    ThreadInit(&threads[i], worker_thread);
    if (ThreadRun(&threads[i], &args[i]) != 0) {
        perror("error while Thread_run");
        num_threads = i;
        break;
    }
}

printf("Поток %d: комбинации %d -%d (всего
%d)\n", i, args[i].start_i,
           args[i].end_i, args[i].end_i - args[i].start_i);
}

for (int i = 0; i < num_threads; i++) {
    if (ThreadJoin(&threads[i]) != 0) {
        perror("error while Thread_join");
    }
}

printf("РЕЗУЛЬТАТЫ\n");
if (best_triangle.area > 0) {
    printf("Максимальная площадь: %.6f\n", best_triangle.area);
}

```

```

    printf("Вершины:\n");
    for (int i = 0; i <3; i++) {
        printf("P%d: (%.6f,%.6f,%.6f)\n",i +
1,best_triangle.vertices[i].x,
best_triangle.vertices[i].y,best_triangle.vertices[i].z);
    }

    free(threads);
    free(args);
    free(points);
    DestroyMutex(&mutex);

    return 0;
}

```

Листинг 1: *Файл задания*

```

#pragma once

#ifndef _WIN32
#else
#include <errno.h>
#include <pthread.h>
#include <stdlib.h>
#endif

typedef pthread_mutex_t mutex_t;

int CreateMutex(mutex_t* mutex);
int LockMutex(mutex_t* mutex);
int UnlockMutex(mutex_t* mutex);
int DestroyMutex(mutex_t* mutex);

```

Листинг 2: *Объявление обёрточных функций для мьютексов*

```

#include "mutex.h"

int CreateMutex(mutex_t* mutex) {
    int res = pthread_mutex_init(mutex,NULL);
    if (res != 0) {
        errno = res;
        return -1;
    }
    return 0;
}

```

```

}

int LockMutex(mutex_t* mutex) {
    int res = pthread_mutex_lock(mutex);
    if (res != 0) {
        errno = res;
        return -1;
    }
    return 0;
}

int UnlockMutex(mutex_t* mutex) {
    int res = pthread_mutex_unlock(mutex);
    if (res != 0) {
        errno = res;
        return -1;
    }
    return 0;
}

int DestroyMutex(mutex_t* mutex) {
    int res = pthread_mutex_destroy(mutex);
    if (res != 0) {
        errno = res;
        return -1;
    }
    return 0;
}

```

Листинг 3: *Реализация мьютексов только для POSIX-систем*

```

#include <errno.h>

#ifndef _WIN32
#else
#include <pthread.h>
#include <unistd.h>
#endif

typedef void* (*ThreadFunc)(void*);

typedef struct {
    ThreadFunc func;
    pthread_t thread;
} Thread;

void ThreadInit(Thread* t, ThreadFunc func);

```

```
int ThreadJoin(Thread* t);

int ThreadDetach(Thread* t);

int ThreadRun(Thread* t,void* arg);
```

Листинг 4: *Объявление обёрточных функций для потоков*

```
#include "thread.h"

void ThreadInit(Thread* t,ThreadFunc func) {
    t->func = func;
    t->thread = 0;
}

int ThreadJoin(Thread* t) {
    int res = pthread_join(t->thread,NULL);
    if (res != 0) {
        errno = res;
    }
    return res;
}

int ThreadDetach(Thread* t) {
    int res = pthread_detach(t->thread);
    if (res != 0) {
        errno = res;
    }
    return res;
}

int ThreadRun(Thread* t,void* arg) {
    int res = pthread_create(&t->thread,NULL,t->func,arg);
    if (res != 0) {
        errno = res;
    }
    return res;
}
```

Листинг 5: *Реализация потоков только для POSIX-систем*

```
#pragma once

#include <math.h>
#include <stdio.h>
```

```

#include <stdlib.h>
#include <string.h>

#ifndef _WIN32
#else
#include <pthread.h>
#include <unistd.h>
#endif

typedef struct {
    double x;
    double y;
    double z;
} Point3D;

typedef struct {
    Point3D vertices[3];
    double area;
} Triangle;

typedef struct {
    int thread_id;
    Point3D* points;
    int num_points;
    int start_i;
    int end_i;
} ThreadArgs;

Point3D CrossProduct(Point3D a,Point3D b);
double VectorMagnitude(Point3D v);
double CalculateTriangleArea(Point3D a,Point3D b,Point3D c);

int ReadPointsFromFile(const char* filename,Point3D** points,int*
num_points);

```

Листинг 6: *Объявление функций для подсчета геометрии*

```

#include "triangle_finder.h"

Point3D CrossProduct(Point3D a,Point3D b) {
    Point3D result;
    result.x = a.y * b.z - a.z * b.y;
    result.y = a.z * b.x - a.x * b.z;
    result.z = a.x * b.y - a.y * b.x;
    return result;
}

```

```

double VectorMagnitude(Point3D v) {
    return sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
}

double CalculateTriangleArea(Point3D a,Point3D b,Point3D c) {
    Point3D ab = {b.x - a.x,b.y - a.y,b.z - a.z};
    Point3D ac = {c.x - a.x,c.y - a.y,c.z - a.z};
    Point3D cross = CrossProduct(ab,ac);
    return 0.5 * VectorMagnitude(cross);
}

int ReadPointsFromFile(const char* filename,Point3D** points,
                      int* num_points) {
    FILE* file = fopen(filename,"r");
    if (!file) {
        perror("error while open file");
        return EXIT_FAILURE;
    }

    unsigned char bom[3];
    size_t n = fread(bom,1,3,file);
    if (n == 3) {
        if (!(bom[0] == 0xEF && bom[1] == 0xBB && bom[2] == 0xBF)) {
            fseek(file,0,SEEK_SET);
        }
    } else {
        fseek(file,0,SEEK_SET);
    }

    *num_points = 0;
    double x,y,z;
    while (fscanf(file,"%lf %lf %lf",&x,&y,&z) == 3) {
        (*num_points)++;
    }

    if (*num_points <3) {
        perror("error: not enough points");
        fclose(file);
        return EXIT_FAILURE;
    }

    *points = malloc(*num_points * sizeof(Point3D));
    if (!*points) {
        perror("error while malloc memmory");
        fclose(file);
        return EXIT_FAILURE;
    }
}

```

```
rewind(file);
for (int i = 0; i < *num_points; i++) {
    fscanf(file, "%lf %lf %lf", &(*points)[i].x, &(*points)[i].y,
           &(*points)[i].z);
}
fclose(file);
printf("Загружено %d точек\n", *num_points);
return 0;
}
```

Листинг 7: *Реализация функций для подсчета геометрии только для POSIX-систем*


```

prlimit64(0,RLIMIT_STACK,NULL,{rlim_cur=8192*1024,rlim_max=RLIM64_INFINITY})
= 0
    munmap(0x7fb14500000,29840) = 0
    getrandom("\xa8\x7a\xf5\x0d\xc7\x47\xdf\x7e",8,GRND_NONBLOCK) = 8
    brk(NULL) = 0x55936a6b8000
    brk(0x55936a6d9000) = 0x55936a6d9000
    openat(AT_FDCWD,"./test_points.txt",O_RDONLY) = 3

newfstatat(3,"",{st_mode=S_IFREG|0644,st_size=121,...},AT_EMPTY_PATH) = 0
    read(3,"0.0 0.0 0.0\n1.0 0.0 0.0\n0.0 1.0 "...,4096) = 121
    lseek(3,0,SEEK_SET) = 0
    read(3,"0.0 0.0 0.0\n1.0 0.0 0.0\n0.0 1.0 "...,4096) = 121
    read(3,"",4096) = 0
    lseek(3,0,SEEK_SET) = 0
    read(3,"0.0 0.0 0.0\n1.0 0.0 0.0\n0.0 1.0 "...,4096) = 121
    read(3,"",4096) = 0
    close(3) = 0

newfstatat(1,"",{st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0x5),...},AT_EMPTY_PATH)
= 0

write(1,"\\320\\227\\320\\260\\320\\263\\321\\200\\321\\203\\320\\266\\320\\265\\320\\275\\320\\276
10 \\321\\202\\320\\276\\321\\207\\320\\265\\320\\272"...,33) = 33
    write(1,"\\320\\222\\321\\201\\320\\265\\320\\263\\320\\276
\\320\\272\\320\\276\\320\\274\\320\\261\\320\\270\\320\\275\\320\\260\\321\\206\\320\\270\\320\\271"...
= 38

rt_sigaction(SIGRT_1,{sa_handler=0x7fb14281870,sa_mask=[],sa_flags=SA_RESTORER|SA_ON
= 0
    rt_sigprocmask(SIG_UNBLOCK,[RTMIN RT_1],NULL,8) = 0

mmap(NULL,8392704,PROT_NONE,MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK,-1,0) =
0x7fb139ec000
    mprotect(0x7fb139ed000,8388608,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|...
=>{parent_tid=[45187]},88) = 45187
    rt_sigprocmask(SIG_SETMASK,[],NULL,8) = 0
    write(1,"\\320\\237\\320\\276\\321\\202\\320\\276\\320\\272 0:
\\320\\272\\320\\276\\320\\274\\320\\261\\320\\270\\320\\275\\320\\260\\321\\206\\320\\270"...,58)
= 58

mmap(NULL,8392704,PROT_NONE,MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK,-1,0) =
0x7fb131eb000
    mprotect(0x7fb131ec000,8388608,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|  
=>{parent_tid=[45188]},88) = 45188  
    rt_sigprocmask(SIG_SETMASK, [],NULL,8) = 0  
    write(1,"\\320\\237\\320\\276\\321\\202\\320\\276\\320\\272 1:  
\\320\\272\\320\\276\\320\\274\\320\\261\\320\\270\\320\\275\\320\\260\\321\\206\\320\\270"...,59)  
= 59  
  
mmap(NULL,8392704,PROT_NONE,MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK,-1,0) =  
0x7fbcb129ea000  
    mprotect(0x7fbcb129eb000,8388608,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|  
=>{parent_tid=[45189]},88) = 45189  
    rt_sigprocmask(SIG_SETMASK, [],NULL,8) = 0  
    futex(0x7fbcb1440ca70,FUTEX_WAIT_PRIVATE,2,NULL) = 0  
    write(1,"\\320\\237\\320\\276\\321\\202\\320\\276\\320\\272 2:  
\\320\\272\\320\\276\\320\\274\\320\\261\\320\\270\\320\\275\\320\\260\\321\\206\\320\\270"...,59)  
= 59  
    futex(0x7fbcb1440ca70,FUTEX_WAKE_PRIVATE,1) = 1  
  
mmap(NULL,8392704,PROT_NONE,MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK,-1,0) =  
0x7fbcb121e9000  
    mprotect(0x7fbcb121ea000,8388608,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|  
=>{parent_tid=[45190]},88) = 45190  
    rt_sigprocmask(SIG_SETMASK, [],NULL,8) = 0  
    write(1,"\\320\\237\\320\\276\\321\\202\\320\\276\\320\\272 3:  
\\320\\272\\320\\276\\320\\274\\320\\261\\320\\270\\320\\275\\320\\260\\321\\206\\320\\270"...,60)  
= 60  
  
futex(0x7fbcb131ea910,FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME,45189,NULL,FUTEX_BITSET_M  
= 0  
  
futex(0x7fbcb129e9910,FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME,45190,NULL,FUTEX_BITSET_M  
= 0  
  
write(1,"\\320\\240\\320\\225\\320\\227\\320\\243\\320\\233\\320\\254\\320\\242\\320\\220\\320\\242\\320  
= 21  
  
write(1,"\\320\\234\\320\\260\\320\\272\\321\\201\\320\\270\\320\\274\\320\\260\\320\\273\\321\\214\\320  
\\320\\277\\320\\273\\320\\276\\321"...,51) = 51  
  
write(1,"\\320\\222\\320\\265\\321\\200\\321\\210\\320\\270\\320\\275\\321\\213:\\n",16)  
= 16  
    write(1,"P1: (1.500000,3.500000,1.00000"...,35) = 35
```

```
write(1,"P2: (-1.000000,-1.000000,0.000"...,37) = 37
write(1,"P3: (2.500000,0.500000,4.00000"...,35) = 35
exit_group(0)                      = ?
+++ exited with 0 +++
```

Листинг 8: *Strace логи*