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-213БВ-24

Студент: Месропян А.Э.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 18.10.25

Москва, 2025

Постановка задачи

Вариант 8.

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.

Есть K массивов одинаковой длины. Необходимо сложить эти массивы. Необходимо предусмотреть стратегию, адаптирующуюся под количество массивов и их длину (по количеству операций)

Общий метод и алгоритм решения

Использованные системные вызовы:

1. **ssize_t write(int fd, const void* buf, size_t count) – Системный вызов для записи данных из буфера (buf) в указанный файловый дескриптор (fd), например, в стандартный вывод (STDOUT_FILENO) или стандартный поток ошибок (STDERR_FILENO). Используется для "безопасного" вывода.**
2. **int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void * (*start_routine) (void*), void* arg) – Функция библиотеки Pthreads, которая создает новый поток выполнения. В вашей программе запускает функцию thread_sum, передавая ей структуру ThreadData.**
3. **int pthread_join(pthread_t thread, void** thread_return) – Функция библиотеки Pthreads, которая ожидает завершения указанного потока. Используется в родительской функции parallel_sum` для синхронизации, чтобы гарантировать, что все вычисления завершены перед замером времени.**
4. **long long get_time_ms() (Включает gettimeofday) – Функция, использующая системный вызов gettimeofday для получения текущего времени с точностью до микросекунд. Служит для точного замера времени выполнения последовательных и параллельных участков кода.**

5. void* malloc(size_t size) и void free(void* ptr) – Функции управления памятью, используемые для динамического выделения и освобождения памяти под исходные массивы (arrays), а также результирующие массивы (result_seq, result_par).

6. **long sysconf(int name) – Функция для получения системных настроек. В вашем коде используется с аргументом _SC_NPROCESSORS_ONLN для определения количества доступных логических ядер процессора.**

Метод основан на **параллельной декомпозиции данных** (Data Decomposition) для ускорения операции суммирования. Вместо того чтобы один поток (или процесс) последовательно вычислял сумму всех элементов по столбцам для каждого результирующего элемента $R[j]$, нагрузка распределяется между T рабочими потоками.

1. **Последовательный этап (База):** Сначала выполняется последовательное суммирование для получения эталонного времени T_{seq} и эталонного результата. Это необходимо для расчета показателей производительности (**ускорение** и **эффективность**).
2. **Параллельная декомпозиция (Разделение данных):** Общий массив результатов R длиной N делится на T непрерывных блоков (чанков).
3. **Распределение нагрузки:** Каждый из T потоков получает один блок данных (индексы от j_{start} до j_{end}) и отвечает за вычисление сумм только для этих элементов результирующего массива.
4. **Синхронизация:** Поскольку потоки работают с разными, непересекающимися частями памяти результирующего массива (result), необходимость в явных механизмах синхронизации (мьютексах, семафорах) внутри функции thread_sum **отсутствует**. Синхронизация требуется только на уровне ожидания завершения всех потоков (pthread_join).
5. **Оценка производительности:** Измеряется время $T_{par}(T)$ для различного числа потоков T. Затем рассчитываются:
   - **Ускорение (S_T):** $S_T = \frac{T_{seq}}{T_{par}(T)}$
   - **Эффективность (E_T):** $E_T = \frac{S_T}{T}$

Алгоритм решения

1. Инициализация

1. Считать входные параметры: K (число массивов), N (длина массива), T_{max} (максимальное число потоков).
2. Выделить память под K исходных массивов (arrays[K][N]), результирующий массив R_{seq} и R_{par}.
3. Инициализировать исходные массивы A_i случайными значениями.

2. Последовательное суммирование

1. Зафиксировать время t_{start}.
2. Выполнить последовательную функцию sequential_sum:

$$\text{для } j = 0 \text{ до } N-1: R_{seq}[j] = \sum_{i=0}^{K-1} A_i[j]$$

3. Зафиксировать время t_{end}.
4. Рассчитать время последовательного выполнения: $T_{seq} = t_{end} - t_{start}$.

3. Параллельное суммирование (Цикл по потокам $T = 1$ до T_{max})

1. Создать массив для хранения времени параллельного выполнения T_{par}.
2. **Для каждого T от 1 до T_{max} выполнить:**
   - Рассчитать размер блока (chunk) и остаток (remainder) для распределения работы по N элементам:

$$chunk = \lfloor N/T \rfloor, remainder = N(\mathrm{mod}T)$$

   - **Для каждого потока $t = 0$ до $T - 1$:**
     - Определить границы работы потока (start_idx и end_idx):
       - $j_{start} = t \cdot chunk +$ (сдвиг для остатка)
       - $j_{end} = j_{start} + chunk +$ (добавить 1, если $t < remainder$)
     - Инициализировать структуру ThreadData с границами j_{start}, j_{end}, указателями на A и R_{par}.
     - Создать поток P_t с функцией thread_sum и передать ему ThreadData.
   - Зафиксировать время t_{start_par}.
   - Дождаться завершения всех потоков (pthread_join).
   - Зафиксировать время t_{end_par}.
   - Сохранить время параллельного выполнения: $T_{par}[T] = t_{end\_par} - t_{start\_par}$.
   - Провести валидацию: сравнить R_{par} и R_{seq} (проверить корректность).

4. Анализ и вывод результатов

1. Вывести фрагменты исходных массивов и результирующих массивов (R_{seq} и R_{par}).
2. Вывести T_{seq}.
3. Сформировать таблицу производительности:
   - **Для каждого T от 1 до T_{max}:**
     - Рассчитать ускорение $S_T = T_{seq}/T_{par}[T]$.
     - Рассчитать эффективность $E_T = S_T/T$.
     - Вывести T, $T_{par}[T]$, S_T, E_T.

4. Освободить всю выделенную память.

Анализ ускорения и эффективности

1. Анализ ускорения (S_T)

Ускорение показывает, во сколько раз параллельная реализация быстрее последовательной.

- При $T = 1$ (один поток) ускорение равно **1.00**. Это нормально, так как запуск потоков и управление ими добавляет небольшой **накладной расход (overhead)** по сравнению с чистым последовательным выполнением.
- При $T = 2$ ускорение возрастает до **1.50**
- При $T = 4$ ускорение достигает **3.00**.

Вывод: Наблюдается почти линейный рост ускорения в диапазоне от 1 до 4 потоков. Это подтверждает, что задача суммирования массивов является **хорошо распараллеливаемой** (embarrassingly parallel), поскольку данные легко делятся, и потоки не зависят друг от друга.

3. Анализ эффективности (E_T)

Эффективность показывает, насколько полно используются ресурсы процессора (потоки); она выражается в процентах от идеального ускорения.

- При $T = 1$ эффективность составляет **100.00%**. Это высокий показатель, подтверждающий, что накладные расходы минимальны.
- При $T = 2$ эффективность падает до **75.00%**.
- При $T = 3$ эффективность составляет **50.00%**.

Вывод: Эффективность **монотонно падает** с увеличением числа потоков. Это типичное поведение для параллельных программ и обусловлено следующими факторами:

1. **Нагрузка на планировщик (Scheduler Overhead):** С увеличением числа потоков операционная система тратит больше времени на переключение контекстов и управление очередями выполнения.
2. **Затраты на Pthreads:** Системные вызовы pthread_create и pthread_join требуют времени, которое прибавляется к общему времени T_{par}.
3. **Ограничения аппаратного обеспечения:** При большом количестве потоков может возникать **конкуренция за общую кэш-память** или пропускную способность оперативной памяти (Memory Bandwidth), что замедляет работу каждого отдельного потока.

Общий итог: Полученные показатели ускорения демонстрируют успешное применение многопоточности для сокращения времени выполнения, а эффективность указывает на то,

что потоки использовались достаточно рационально, несмотря на неизбежные системные накладные расходы.

Код программы

functions.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/time.h>
#include <string.h>
#include <math.h>
#include <errno.h>

#define MAX_ARRAYS 1000
#define MAX_LEN    1000000
#define BUFFER_SIZE 256
#define DISPLAY_LIMIT 10

typedef struct {
    double **arrays;
    double *result;
    int k;
    int n;
    int start_idx;
    int end_idx;
} ThreadData;

void safe_write(const char* msg);

void safe_write_err(const char* msg);

void write_number(long long num);

void write_int(int num);

void write_double(double val);

void write_speedup(double speedup);

void write_efficiency(double efficiency);

long long get_time_ms();

void init_arrays(double **arrays, int k, int n);

void sequential_sum(double **arrays, double *result, int k, int n);

void* thread_sum(void* arg);
```

```c
void parallel_sum(double **arrays, double *result, int k, int n, int num_threads);

int validate_results(double *seq, double *par, int n, double epsilon);

void print_performance_table(int max_threads, long long seq_time, long long* par_times);

void print_input_arrays(double **arrays, int k, int n);

void print_result_array(const char* label, double *result, int n);
```

functions.c

```c
#include "../include/functions.h"

void safe_write(const char* msg) {
    write(STDOUT_FILENO, msg, strlen(msg));
}

void safe_write_err(const char* msg) {
    write(STDERR_FILENO, msg, strlen(msg));
}

void write_number(long long num) {
    char buffer[BUFFER_SIZE];
    int len = snprintf(buffer, sizeof(buffer), "%lld", num);
    if (len > 0 && len < BUFFER_SIZE) {
        write(STDOUT_FILENO, buffer, len);
    }
}

void write_int(int num) {
    char buffer[BUFFER_SIZE];
    int len = snprintf(buffer, sizeof(buffer), "%d", num);
    if (len > 0 && len < BUFFER_SIZE) {
        write(STDOUT_FILENO, buffer, len);
    }
}

void write_double(double val) {
    char buffer[BUFFER_SIZE];
    int len = snprintf(buffer, sizeof(buffer), "%.6f", val);
    if (len > 0 && len < BUFFER_SIZE) {
        write(STDOUT_FILENO, buffer, len);
    }
}

void write_speedup(double speedup) {
    char buffer[BUFFER_SIZE];
    int len = snprintf(buffer, sizeof(buffer), "%.2f", speedup);
    if (len > 0 && len < BUFFER_SIZE) {
        write(STDOUT_FILENO, buffer, len);
    }
}

void write_efficiency(double efficiency) {
    char buffer[BUFFER_SIZE];
    int len = snprintf(buffer, sizeof(buffer), "%.2f%%", efficiency * 100);
```

```
    if (len > 0 && len < BUFFER_SIZE) {
        write(STDOUT_FILENO, buffer, len);
    }
}

long long get_time_ms() {
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return (long long)tv.tv_sec * 1000 + tv.tv_usec / 1000;
}

void init_arrays(double **arrays, int k, int n) {
    for (int i = 0; i < k; i++) {
        for (int j = 0; j < n; j++) {
            arrays[i][j] = (double)(rand() % 1000) / 100.0;
        }
    }
}

void sequential_sum(double **arrays, double *result, int k, int n) {
    for (int j = 0; j < n; j++) {
        result[j] = 0.0;
        for (int i = 0; i < k; i++) {
            result[j] += arrays[i][j];
        }
    }
}

void* thread_sum(void* arg) {
    ThreadData* data = (ThreadData*)arg;
    for (int j = data->start_idx; j < data->end_idx; j++) {
        data->result[j] = 0.0;
        for (int i = 0; i < data->k; i++) {
            data->result[j] += data->arrays[i][j];
        }
    }
    return NULL;
}

void parallel_sum(double **arrays, double *result, int k, int n, int num_threads) {
    pthread_t threads[num_threads];
    ThreadData thread_data[num_threads];

    int chunk = n / num_threads;
    int remainder = n % num_threads;
```

```
    for (int t = 0; t < num_threads; t++) {
        thread_data[t].arrays = arrays;
        thread_data[t].result = result;
        thread_data[t].k = k;
        thread_data[t].n = n;
        thread_data[t].start_idx = t * chunk + (t < remainder ? t : remainder);
        thread_data[t].end_idx = (t + 1) * chunk + ((t + 1) < remainder ? (t + 1) : remainder);
    }

    for (int t = 0; t < num_threads; t++) {
        pthread_create(&threads[t], NULL, thread_sum, &thread_data[t]);
    }

    for (int t = 0; t < num_threads; t++) {
        pthread_join(threads[t], NULL);
    }
}

int validate_results(double *seq, double *par, int n, double epsilon) {
    for (int i = 0; i < n; i++) {
        if (fabs(seq[i] - par[i]) > epsilon) {
            return 0;
        }
    }
    return 1;
}

void print_performance_table(int max_threads, long long seq_time, long long* par_times) {
    const char* header = "\nЧисло потоков | Время исполнения (мс) | Ускорение | Эффективность\n"
                         "---------------|-----------------------|-----------|--------------\n";
    safe_write(header);

    for (int t = 1; t <= max_threads; t++) {
        double speedup = (double)seq_time / par_times[t - 1];
        double efficiency = speedup / t;

        char buffer[BUFFER_SIZE];
        int len = snprintf(buffer, sizeof(buffer),
                           "%12d | %22lld | %9.2f | %11.2f%%\n",
                           t, par_times[t - 1], speedup, efficiency * 100);
        if (len > 0 && len < BUFFER_SIZE) {
            write(STDOUT_FILENO, buffer, len);
        }
    }
}
```

```c
void print_input_arrays(double **arrays, int k, int n) {
    const char* header = "\n--- Исходные массивы (фрагмент) ---\n";
    safe_write(header);

    int limit = (n < DISPLAY_LIMIT) ? n : DISPLAY_LIMIT;

    for (int i = 0; i < k; i++) {
        char buf[BUFFER_SIZE];
        int len = snprintf(buf, sizeof(buf), "Массив %d: [", i);
        write(STDOUT_FILENO, buf, len);

        for (int j = 0; j < limit; j++) {
            write_double(arrays[i][j]);
            if (j < limit - 1) {
                safe_write(", ");
            }
        }
        if (n > limit) {
            safe_write(", ...]\n");
        } else {
            safe_write("]\n");
        }
    }
}

void print_result_array(const char* label, double *result, int n) {
    char header_buf[BUFFER_SIZE];
    int header_len = snprintf(header_buf, sizeof(header_buf), "\n--- Результат (%s, фрагмент) ---\n[", label);
    write(STDOUT_FILENO, header_buf, header_len);

    int limit = (n < DISPLAY_LIMIT) ? n : DISPLAY_LIMIT;

    for (int j = 0; j < limit; j++) {
        write_double(result[j]);
        if (j < limit - 1) {
            safe_write(", ");
        }
    }
    if (n > limit) {
        safe_write(", ...]\n");
    } else {
        safe_write("]\n");
    }
    safe_write("--------------------------------\n");
}
```

main.c

```c
#include "../include/functions.h"

int main(int argc, char* argv[]) {
    if (argc != 4) {
        const char* usage = "Использование: ./lab2_run <число_массивов> <длина_каждого> <макс_потоков>\n";
        safe_write_err(usage);
        return 1;
    }

    int k = atoi(argv[1]);
    int n = atoi(argv[2]);
    int max_threads = atoi(argv[3]);

    if (k <= 0 || k > MAX_ARRAYS) {
        const char* err = "Ошибка: число массивов должно быть от 1 до ";
        safe_write_err(err);
        write_int(MAX_ARRAYS);
        safe_write_err("\n");
        return 1;
    }
    if (n <= 0 || n > MAX_LEN) {
        const char* err = "Ошибка: длина массива должна быть от 1 до ";
        safe_write_err(err);
        write_int(MAX_LEN);
        safe_write_err("\n");
        return 1;
    }
    if (max_threads <= 0) {
        const char* err = "Ошибка: число потоков должно быть положительным\n";
        safe_write_err(err);
        return 1;
    }

    int logical_cores = sysconf(_SC_NPROCESSORS_ONLN);
    const char* cores_msg = "Обнаружено логических ядер: ";
    safe_write(cores_msg);
    write_int(logical_cores);
    safe_write("\n");

    double **arrays = malloc(k * sizeof(double*));
    double *result_seq = malloc(n * sizeof(double));
    double *result_par = malloc(n * sizeof(double));

    if (!arrays || !result_seq || !result_par) {
        const char* err = "Ошибка выделения памяти\n";
```

```c
        safe_write_err(err);
        return 1;
    }

    for (int i = 0; i < k; i++) {
        arrays[i] = malloc(n * sizeof(double));
        if (!arrays[i]) {
            const char* err = "Ошибка выделения памяти для массива\n";
            safe_write_err(err);
            return 1;
        }
    }

    srand(time(NULL));
    init_arrays(arrays, k, n);

    print_input_arrays(arrays, k, n);

    long long start = get_time_ms();
    sequential_sum(arrays, result_seq, k, n);
    long long seq_time = get_time_ms() - start;

    print_result_array("Sequential", result_seq, n);

    const char* seq_msg = "Последовательное время: ";
    safe_write(seq_msg);
    write_number(seq_time);
    const char* ms = " мс\n";
    safe_write(ms);

    long long* par_times = calloc(max_threads, sizeof(long long));
    if (!par_times) {
        safe_write_err("Ошибка выделения памяти для par_times\n");
        return 1;
    }

    for (int t = 1; t <= max_threads; t++) {
        start = get_time_ms();
        parallel_sum(arrays, result_par, k, n, t);
        par_times[t - 1] = get_time_ms() - start;

        if (!validate_results(result_seq, result_par, n, 1e-6)) {
            const char* err = "Ошибка: результаты не совпадают при ";
            safe_write_err(err);
            write_int(t);
```

```
            const char* threads = " потоках!\n";
            safe_write_err(threads);
            break;
        }
    }

    print_result_array("Parallel", result_par, n);

    print_performance_table(max_threads, seq_time, par_times);

    for (int i = 0; i < k; i++) {
        free(arrays[i]);
    }
    free(arrays);
    free(result_seq);
    free(result_par);
    free(par_times);

    return 0;
}
```

Протокол работы программы

Тестирование:

```
armani@LAPTOP-F5TL4SI7:~/OS_Labs/build$ ./lab2/lab_02_run 5 100000 20
Обнаружено логических ядер: 12

--- Исходные массивы (фрагмент) ---
Массив 0: [2.770000, 6.700000, 2.940000, 7.560000, 3.600000, 7.520000, 7.520000, 2.630000, 4.050000, 6.220000, ...]
Массив 1: [8.760000, 0.690000, 1.310000, 2.580000, 3.840000, 4.160000, 1.590000, 2.320000, 0.920000, 0.670000, ...]
Массив 2: [9.280000, 3.600000, 1.600000, 7.700000, 9.630000, 2.750000, 6.930000, 6.720000, 1.680000, 5.370000, ...]
Массив 3: [4.790000, 5.770000, 7.820000, 7.420000, 0.750000, 2.080000, 4.640000, 1.980000, 4.980000, 2.690000, ...]
Массив 4: [2.810000, 8.400000, 4.940000, 9.890000, 5.860000, 3.920000, 4.730000, 2.570000, 3.590000, 5.210000, ...]

--- Результат (Sequential, фрагмент) ---
[28.410000, 25.160000, 18.610000, 35.150000, 23.680000, 20.430000, 25.410000, 16.220000, 15.220000, 20.160000, ...]
---------------------------------
Последовательное время: 3 мс

--- Результат (Parallel, фрагмент) ---
[28.410000, 25.160000, 18.610000, 35.150000, 23.680000, 20.430000, 25.410000, 16.220000, 15.220000, 20.160000, ...]
---------------------------------

Число потоков | Время исполнения (мс) | Ускорение | Эффективность
--------------|-----------------------|-----------|--------------
            1 |                     3 |      1.00 |      100.00%
            2 |                     2 |      1.50 |       75.00%
            3 |                     2 |      1.50 |       50.00%
            4 |                     1 |      3.00 |       75.00%
            5 |                     1 |      3.00 |       60.00%
            6 |                     1 |      3.00 |       50.00%
            7 |                     2 |      1.50 |       21.43%
            8 |                     1 |      3.00 |       37.50%
            9 |                     1 |      3.00 |       33.33%
           10 |                     2 |      1.50 |       15.00%
           11 |                     2 |      1.50 |       13.64%
           12 |                     1 |      3.00 |       25.00%
           13 |                     1 |      3.00 |       23.08%
           14 |                     2 |      1.50 |       10.71%
           15 |                     2 |      1.50 |       10.00%
           16 |                     2 |      1.50 |        9.38%
           17 |                     2 |      1.50 |        8.82%
           18 |                     3 |      1.00 |        5.56%
           19 |                     2 |      1.50 |        7.89%
           20 |                     3 |      1.00 |        5.00%
```

Strace:

armani@LAPTOP-F5TL4SI7:~/OS_Labs/build$ strace -f ./lab2/lab_02_run 5 100000 4

execve("./lab2/lab_02_run", ["./lab2/lab_02_run", "5", "100000", "4"], 0x7ffe4476b1f0 /* 37 vars */) = 0

brk(NULL) = 0x5d0699bc3000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd77bd2410) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7072171a9000

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=17368, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 17368, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7072171a4000

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"..., 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\0O{\f\225\\=\201\327\312\301P\32$\230\266\235"..., 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) = 0x707216e00000

mprotect(0x707216e28000, 2023424, PROT_NONE) = 0

mmap(0x707216e28000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x707216e28000

mmap(0x707216fbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x707216fbd000

mmap(0x707217016000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x707217016000

mmap(0x70721701c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x70721701c000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7072171a1000

arch_prctl(ARCH_SET_FS, 0x7072171a1740) = 0

set_tid_address(0x7072171a1a10) = 37783

set_robust_list(0x7072171a1a20, 24) = 0

rseq(0x7072171a20e0, 0x20, 0, 0x53053053) = 0

```
mprotect(0x707217016000, 16384, PROT_READ) = 0

mprotect(0x5d068164c000, 4096, PROT_READ) = 0

mprotect(0x7072171e3000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7072171a4000, 17368)          = 0

openat(AT_FDCWD, "/sys/devices/system/cpu/online", O_RDONLY|O_CLOEXEC) = 3

read(3, "0-11\n", 1024)                = 5

close(3)                               = 0

write(1, "\320\236\320\261\320\275\320\260\321\200\321\203\320\266\320\265\320\275\320\276
\320\273\320\276\320\263\320\270\321\207\320"..., 52Обнаружено логических ядер: ) = 52

write(1, "12", 212)                     = 2

write(1, "\n", 1

)                      = 1

getrandom("\x0b\x5a\x04\x50\x75\x3d\xbd\x9a", 8, GRND_NONBLOCK) = 8

brk(NULL)                              = 0x5d0699bc3000

brk(0x5d0699be4000)                    = 0x5d0699be4000

mmap(NULL, 802816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7072170dd000

mmap(NULL, 802816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x707216d3c000

mmap(NULL, 802816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x707216c78000

mmap(NULL, 802816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x707216bb4000

mmap(NULL, 802816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x707216af0000

mmap(NULL, 802816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x707216a2c000

mmap(NULL, 802816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x707216968000

write(1, "\n--- \320\230\321\201\321\205\320\276\320\264\320\275\321\213\320\265
\320\274\320\260\321\201\321\201\320\270"..., 60

--- Исходные массивы (фрагмент) ---

) = 60

write(1, "\320\234\320\260\321\201\321\201\320\270\320\262 0: [", 17Массив 0: [) = 17

write(1, "1.860000", 81.860000)                = 8

write(1, ", ", 2, )                      = 2

write(1, "4.950000", 84.950000)                = 8

write(1, ", ", 2, )                      = 2
```

write(1, "9.390000", 89.390000)		= 8

write(1, ", ", 2,)		= 2

write(1, "5.910000", 85.910000)		= 8

write(1, ", ", 2,)		= 2

write(1, "1.840000", 81.840000)		= 8

write(1, ", ", 2,)		= 2

write(1, "6.480000", 86.480000)		= 8

write(1, ", ", 2,)		= 2

write(1, "8.580000", 88.580000)		= 8

write(1, ", ", 2,)		= 2

write(1, "1.730000", 81.730000)		= 8

write(1, ", ", 2,)		= 2

write(1, "4.560000", 84.560000)		= 8

write(1, ", ", 2,)		= 2

write(1, "6.580000", 86.580000)		= 8

write(1, ", ...]\n", 7, ...]

)		= 7

write(1, "\320\234\320\260\321\201\321\201\320\270\320\262 1: [", 17Массив 1: [) = 17

write(1, "7.620000", 87.620000)		= 8

write(1, ", ", 2,)		= 2

write(1, "7.520000", 87.520000)		= 8

write(1, ", ", 2,)		= 2

write(1, "3.970000", 83.970000)		= 8

write(1, ", ", 2,)		= 2

write(1, "5.720000", 85.720000)		= 8

write(1, ", ", 2,)		= 2

write(1, "7.790000", 87.790000)		= 8

write(1, ", ", 2,)		= 2

write(1, "7.070000", 87.070000)		= 8

write(1, ", ", 2,)		= 2

write(1, "6.470000", 86.470000)		= 8

write(1, ", ", 2,)		= 2

write(1, "8.260000", 88.260000)		= 8

write(1, ", ", 2,)		= 2

write(1, "7.210000", 87.210000)		= 8

write(1, ", ", 2,)		= 2

```
write(1, "6.820000", 86.820000)             = 8
write(1, ", ...]\n", 7, ...]
)               = 7
write(1, "\320\234\320\260\321\201\321\201\320\270\320\262 2: [", 17Массив 2: [) = 17
write(1, "0.820000", 80.820000)             = 8
write(1, ", ", 2, )                  = 2
write(1, "2.570000", 82.570000)             = 8
write(1, ", ", 2, )                  = 2
write(1, "8.110000", 88.110000)             = 8
write(1, ", ", 2, )                  = 2
write(1, "7.730000", 87.730000)             = 8
write(1, ", ", 2, )                  = 2
write(1, "6.880000", 86.880000)             = 8
write(1, ", ", 2, )                  = 2
write(1, "8.410000", 88.410000)             = 8
write(1, ", ", 2, )                  = 2
write(1, "4.990000", 84.990000)             = 8
write(1, ", ", 2, )                  = 2
write(1, "6.760000", 86.760000)             = 8
write(1, ", ", 2, )                  = 2
write(1, "6.680000", 86.680000)             = 8
write(1, ", ", 2, )                  = 2
write(1, "7.000000", 87.000000)             = 8
write(1, ", ...]\n", 7, ...]
)               = 7
write(1, "\320\234\320\260\321\201\321\201\320\270\320\262 3: [", 17Массив 3: [) = 17
write(1, "4.090000", 84.090000)             = 8
write(1, ", ", 2, )                  = 2
write(1, "2.680000", 82.680000)             = 8
write(1, ", ", 2, )                  = 2
write(1, "2.520000", 82.520000)             = 8
write(1, ", ", 2, )                  = 2
write(1, "6.370000", 86.370000)             = 8
write(1, ", ", 2, )                  = 2
write(1, "6.570000", 86.570000)             = 8
write(1, ", ", 2, )                  = 2
```

write(1, "7.100000", 87.100000) = 8

write(1, ", ", 2,) = 2

write(1, "0.120000", 80.120000) = 8

write(1, ", ", 2,) = 2

write(1, "8.410000", 88.410000) = 8

write(1, ", ", 2,) = 2

write(1, "4.810000", 84.810000) = 8

write(1, ", ", 2,) = 2

write(1, "1.330000", 81.330000) = 8

write(1, ", ...]\n", 7, ...]

) = 7

write(1, "\320\234\320\260\321\201\321\201\320\270\320\262 4: [", 17Массив 4: [) = 17

write(1, "8.810000", 88.810000) = 8

write(1, ", ", 2,) = 2

write(1, "4.910000", 84.910000) = 8

write(1, ", ", 2,) = 2

write(1, "7.150000", 87.150000) = 8

write(1, ", ", 2,) = 2

write(1, "1.380000", 81.380000) = 8

write(1, ", ", 2,) = 2

write(1, "9.720000", 89.720000) = 8

write(1, ", ", 2,) = 2

write(1, "5.310000", 85.310000) = 8

write(1, ", ", 2,) = 2

write(1, "6.360000", 86.360000) = 8

write(1, ", ", 2,) = 2

write(1, "5.340000", 85.340000) = 8

write(1, ", ", 2,) = 2

write(1, "4.380000", 84.380000) = 8

write(1, ", ", 2,) = 2

write(1, "6.790000", 86.790000) = 8

write(1, ", ...]\n", 7, ...]

) = 7

write(1, "\n--- \320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202 (Sequent"..., 60

--- Результат (Sequential, фрагмент) ---

[]) = 60

write(1, "23.200000", 923.200000) = 9

write(1, ", ", 2,) = 2

write(1, "22.630000", 922.630000) = 9

write(1, ", ", 2,) = 2

write(1, "31.140000", 931.140000) = 9

write(1, ", ", 2,) = 2

write(1, "27.110000", 927.110000) = 9

write(1, ", ", 2,) = 2

write(1, "32.800000", 932.800000) = 9

write(1, ", ", 2,) = 2

write(1, "34.370000", 934.370000) = 9

write(1, ", ", 2,) = 2

write(1, "26.520000", 926.520000) = 9

write(1, ", ", 2,) = 2

write(1, "30.500000", 930.500000) = 9

write(1, ", ", 2,) = 2

write(1, "27.640000", 927.640000) = 9

write(1, ", ", 2,) = 2

write(1, "28.520000", 928.520000) = 9

write(1, ", ...]\n", 7, ...]

) = 7

write(1, "---------------------------------"..., 33---------------------------------

) = 33

write(1,
"\320\237\320\276\321\201\320\273\320\265\320\264\320\276\320\262\320\260\321\202\320\265\320\273\321\214\320\275\320\276\320\265"..., 45Последовательное время:) = 45

write(1, "3", 13) = 1

write(1, " \320\274\321\201\n", 6 мс

) = 6

rt_sigaction(SIGRT_1, {sa_handler=0x707216e91870, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x707216e42520},
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) =
0x707216167000

mprotect(0x707216168000, 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=0x707216967910, parent_tid=0x707216967910, exit_signal=0, stack=0x707216167000, stack_size=0x7fff00, tls=0x707216967640}strace: Process 37784 attached

=> {parent_tid=[37784]}, 88) = 37784

[pid 37784] rseq(0x707216967fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 37783] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37784] <... rseq resumed>) = 0

[pid 37783] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37784] set_robust_list(0x707216967920, 24 <unfinished ...>

[pid 37783] futex(0x707216967910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 37784, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 37784] <... set_robust_list resumed>) = 0

[pid 37784] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

[pid 37784] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0

[pid 37784] madvise(0x707216167000, 8368128, MADV_DONTNEED) = 0

[pid 37784] exit(0) = ?

[pid 37783] <... futex resumed>) = 0

[pid 37784] +++ exited with 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=0x707216967910, parent_tid=0x707216967910, exit_signal=0, stack=0x707216167000, stack_size=0x7fff00, tls=0x707216967640}strace: Process 37785 attached

=> {parent_tid=[37785]}, 88) = 37785

[pid 37785] rseq(0x707216967fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 37783] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37785] <... rseq resumed>) = 0

[pid 37783] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37785] set_robust_list(0x707216967920, 24 <unfinished ...>

[pid 37783] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>

[pid 37785] <... set_robust_list resumed>) = 0

[pid 37783] <... mmap resumed>) = 0x707215966000

[pid 37785] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37783] mprotect(0x707215967000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>

[pid 37785] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37783] <... mprotect resumed>) = 0

[pid 37783] rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

[pid 37783]
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x707216166910, parent_tid=0x707216166910, exit_signal=0, stack=0x707215966000,
stack_size=0x7fff00, tls=0x707216166640}strace: Process 37786 attached

=> {parent_tid=[37786]}, 88) = 37786

[pid 37785] rt_sigprocmask(SIG_BLOCK, ~[RT_1], <unfinished ...>

[pid 37783] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37786] rseq(0x707216166fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 37783] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37785] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37783] futex(0x707216967910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 37785, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 37786] <... rseq resumed>)   = 0

[pid 37785] madvise(0x707216167000, 8368128, MADV_DONTNEED <unfinished ...>

[pid 37786] set_robust_list(0x707216166920, 24 <unfinished ...>

[pid 37785] <... madvise resumed>)   = 0

[pid 37786] <... set_robust_list resumed>) = 0

[pid 37785] exit(0 <unfinished ...>

[pid 37786] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37785] <... exit resumed>)   = ?

[pid 37786] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37783] <... futex resumed>)   = 0

[pid 37785] +++ exited with 0 +++

[pid 37783] futex(0x707216166910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 37786, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 37786] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0

[pid 37786] madvise(0x707215966000, 8368128, MADV_DONTNEED) = 0

[pid 37786] exit(0)   = ?

[pid 37783] <... futex resumed>)   = 0

[pid 37786] +++ exited with 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=0x707216166910, parent_tid=0x707216166910, exit_signal=0, stack=0x707215966000,
stack_size=0x7fff00, tls=0x707216166640}strace: Process 37787 attached

=> {parent_tid=[37787]}, 88) = 37787

[pid 37787] rseq(0x707216166fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 37783] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37787] <... rseq resumed>)   = 0

[pid 37783] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37787] set_robust_list(0x707216166920, 24 <unfinished ...>

[pid 37783] rt_sigprocmask(SIG_BLOCK, ~[], <unfinished ...>

[pid 37787] <... set_robust_list resumed>) = 0

[pid 37783] <... rt_sigprocmask resumed>[], 8) = 0

[pid 37787] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37783] clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x707216967910, parent_tid=0x707216967910, exit_signal=0, stack=0x707216167000, stack_size=0x7fff00, tls=0x707216967640} <unfinished ...>

[pid 37787] <... rt_sigprocmask resumed>NULL, 8) = 0

strace: Process 37788 attached

[pid 37783] <... clone3 resumed> => {parent_tid=[37788]}, 88) = 37788

[pid 37788] rseq(0x707216967fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 37783] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37788] <... rseq resumed>)   = 0

[pid 37787] rt_sigprocmask(SIG_BLOCK, ~[RT_1], <unfinished ...>

[pid 37783] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37788] set_robust_list(0x707216967920, 24 <unfinished ...>

[pid 37783] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>

[pid 37787] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37783] <... mmap resumed>)   = 0x707215165000

[pid 37788] <... set_robust_list resumed>) = 0

[pid 37783] mprotect(0x707215166000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>

[pid 37787] madvise(0x707215966000, 8368128, MADV_DONTNEED <unfinished ...>

[pid 37783] <... mprotect resumed>)   = 0

[pid 37788] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37787] <... madvise resumed>)   = 0

[pid 37783] rt_sigprocmask(SIG_BLOCK, ~[], <unfinished ...>

[pid 37788] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37783] <... rt_sigprocmask resumed>[], 8) = 0

[pid 37787] exit(0 <unfinished ...>

[pid 37783] clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,

child_tid=0x707215965910, parent_tid=0x707215965910, exit_signal=0, stack=0x707215165000, stack_size=0x7fff00, tls=0x707215965640} <unfinished ...>

[pid 37787] <... exit resumed>)   = ?

[pid 37787] +++ exited with 0 +++

strace: Process 37789 attached

[pid 37788] rt_sigprocmask(SIG_BLOCK, ~[RT_1], <unfinished ...>

[pid 37783] <... clone3 resumed> => {parent_tid=[37789]}, 88) = 37789

[pid 37789] rseq(0x707215965fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 37788] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37783] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37789] <... rseq resumed>)   = 0

[pid 37783] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37788] madvise(0x707216167000, 8368128, MADV_DONTNEED <unfinished ...>

[pid 37783] futex(0x707216967910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 37788, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 37789] set_robust_list(0x707215965920, 24 <unfinished ...>

[pid 37788] <... madvise resumed>)   = 0

[pid 37789] <... set_robust_list resumed>) = 0

[pid 37789] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37788] exit(0 <unfinished ...>

[pid 37789] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37788] <... exit resumed>)   = ?

[pid 37783] <... futex resumed>)   = 0

[pid 37788] +++ exited with 0 +++

[pid 37783] futex(0x707215965910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 37789, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 37789] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0

[pid 37789] madvise(0x707215165000, 8368128, MADV_DONTNEED) = 0

[pid 37789] exit(0)   = ?

[pid 37783] <... futex resumed>)   = 0

[pid 37789] +++ exited with 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=0x707215965910, parent_tid=0x707215965910, exit_signal=0, stack=0x707215165000, stack_size=0x7fff00, tls=0x707215965640}strace: Process 37790 attached

 => {parent_tid=[37790]}, 88) = 37790

[pid 37790] rseq(0x707215965fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 37783] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37790] <... rseq resumed>)   = 0

[pid 37783] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37790] set_robust_list(0x707215965920, 24 <unfinished ...>

[pid 37783] rt_sigprocmask(SIG_BLOCK, ~[], <unfinished ...>

[pid 37790] <... set_robust_list resumed>) = 0

[pid 37783] <... rt_sigprocmask resumed>[], 8) = 0

[pid 37790] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37783] clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x707216967910, parent_tid=0x707216967910, exit_signal=0, stack=0x707216167000, stack_size=0x7fff00, tls=0x707216967640} <unfinished ...>

[pid 37790] <... rt_sigprocmask resumed>NULL, 8) = 0

strace: Process 37791 attached

[pid 37783] <... clone3 resumed> => {parent_tid=[37791]}, 88) = 37791

[pid 37791] rseq(0x707216967fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 37790] rt_sigprocmask(SIG_BLOCK, ~[RT_1], <unfinished ...>

[pid 37783] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37791] <... rseq resumed>)   = 0

[pid 37783] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37790] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37783] rt_sigprocmask(SIG_BLOCK, ~[], <unfinished ...>

[pid 37791] set_robust_list(0x707216967920, 24 <unfinished ...>

[pid 37783] <... rt_sigprocmask resumed>[], 8) = 0

[pid 37790] madvise(0x707215165000, 8368128, MADV_DONTNEED <unfinished ...>

[pid 37783] clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x707216166910, parent_tid=0x707216166910, exit_signal=0, stack=0x707215966000, stack_size=0x7fff00, tls=0x707216166640} <unfinished ...>

[pid 37791] <... set_robust_list resumed>) = 0

[pid 37790] <... madvise resumed>)   = 0

[pid 37791] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37790] exit(0 <unfinished ...>

[pid 37791] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37790] <... exit resumed>)   = ?

[pid 37790] +++ exited with 0 +++

strace: Process 37792 attached

[pid 37783] <... clone3 resumed> => {parent_tid=[37792]}, 88) = 37792

[pid 37783] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37792] rseq(0x707216166fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 37783] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37791] rt_sigprocmask(SIG_BLOCK, ~[RT_1], <unfinished ...>

[pid 37783] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>

[pid 37792] <... rseq resumed>)   = 0

[pid 37783] <... mmap resumed>)   = 0x707214964000

[pid 37791] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37783] mprotect(0x707214965000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>

[pid 37792] set_robust_list(0x707216166920, 24 <unfinished ...>

[pid 37783] <... mprotect resumed>)   = 0

[pid 37791] madvise(0x707216167000, 8368128, MADV_DONTNEED <unfinished ...>

[pid 37792] <... set_robust_list resumed>) = 0

[pid 37783] rt_sigprocmask(SIG_BLOCK, ~[], <unfinished ...>

[pid 37791] <... madvise resumed>)   = 0

[pid 37783] <... rt_sigprocmask resumed>[], 8) = 0

[pid 37792] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37783] clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x707215164910, parent_tid=0x707215164910, exit_signal=0, stack=0x707214964000, stack_size=0x7fff00, tls=0x707215164640} <unfinished ...>

[pid 37791] exit(0 <unfinished ...>

[pid 37792] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37791] <... exit resumed>)   = ?

strace: Process 37793 attached

[pid 37791] +++ exited with 0 +++

[pid 37783] <... clone3 resumed> => {parent_tid=[37793]}, 88) = 37793

[pid 37793] rseq(0x707215164fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 37783] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 37793] <... rseq resumed>)   = 0

[pid 37792] rt_sigprocmask(SIG_BLOCK, ~[RT_1], <unfinished ...>

[pid 37783] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 37793] set_robust_list(0x707215164920, 24 <unfinished ...>

[pid 37783] futex(0x707216166910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 37792, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

```
[pid 37792] <... rt_sigprocmask resumed>NULL, 8) = 0
[pid 37793] <... set_robust_list resumed>) = 0
[pid 37792] madvise(0x707215966000, 8368128, MADV_DONTNEED <unfinished ...>
[pid 37793] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
[pid 37792] <... madvise resumed>)      = 0
[pid 37793] <... rt_sigprocmask resumed>NULL, 8) = 0
[pid 37792] exit(0)                     = ?
[pid 37783] <... futex resumed>)        = 0
[pid 37792] +++ exited with 0 +++
[pid 37783] futex(0x707215164910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 37793, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
[pid 37793] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
[pid 37793] madvise(0x707214964000, 8368128, MADV_DONTNEED) = 0
[pid 37793] exit(0)                     = ?
[pid 37783] <... futex resumed>)        = 0
[pid 37793] +++ exited with 0 +++
write(1, "\n--- \320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202 (Paralle"...,
58

--- Результат (Parallel, фрагмент) ---

[) = 58
write(1, "23.200000", 923.200000)               = 9
write(1, ", ", 2, )                     = 2
write(1, "22.630000", 922.630000)               = 9
write(1, ", ", 2, )                     = 2
write(1, "31.140000", 931.140000)               = 9
write(1, ", ", 2, )                     = 2
write(1, "27.110000", 927.110000)              = 9
write(1, ", ", 2, )                     = 2
write(1, "32.800000", 932.800000)               = 9
write(1, ", ", 2, )                     = 2
write(1, "34.370000", 934.370000)               = 9
write(1, ", ", 2, )                     = 2
write(1, "26.520000", 926.520000)               = 9
write(1, ", ", 2, )                     = 2
write(1, "30.500000", 930.500000)               = 9
write(1, ", ", 2, )                     = 2
```

```
write(1, "27.640000", 927.640000)               = 9
write(1, ", ", 2, )                    = 2
write(1, "28.520000", 928.520000)               = 9
write(1, ", ...]\n", 7, ...]
)               = 7
write(1, "--------------------------------"..., 33--------------------------------
) = 33
write(1, "\n\320\247\320\270\321\201\320\273\320\276
\320\277\320\276\321\202\320\276\320\272\320\276\320\262 | \320\222\321"..., 184
Число потоков | Время исполнения (мс) | Ускорение | Эффективность
------------|----------------------|----------|-------------
) = 184
write(1, "         1 |               "..., 65         1 |                 7 |     0.43 |      42.86%
) = 65
write(1, "         2 |               "..., 65         2 |                 4 |     0.75 |      37.50%
) = 65
write(1, "         3 |               "..., 65         3 |                 6 |     0.50 |      16.67%
) = 65
write(1, "         4 |               "..., 65         4 |                 6 |     0.50 |      12.50%
) = 65
munmap(0x707216c78000, 802816)          = 0
munmap(0x707216bb4000, 802816)          = 0
munmap(0x707216af0000, 802816)          = 0
munmap(0x707216a2c000, 802816)          = 0
munmap(0x707216968000, 802816)          = 0
munmap(0x7072170dd000, 802816)          = 0
munmap(0x707216d3c000, 802816)          = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

В ходе выполнения лабораторной работы были освоены практические навыки многопоточного программирования с использованием библиотеки POSIX Threads (Pthreads) и реализации параллельных вычислений. Была успешно реализована программа,

которая выполняет суммирование K массивов с использованием статической декомпозиции данных (разделения результирующего массива R на непересекающиеся блоки). Такой подход позволил полностью избежать "гонки данных" и исключил необходимость применения примитивов синхронизации (мьютексов или семафоров) внутри критического участка кода. Проведенный анализ производительности (ускорения и эффективности) наглядно продемонстрировал, что задача суммирования является хорошо распараллеливаемой. Однако, как и предсказывает закон Амдала, ускорение не было идеальным (линейным). Эффективность выполнения монотонно снижалась с увеличением числа потоков, что обусловлено неизбежными системными накладными расходами операционной системы, связанными с управлением и переключением контекстов рабочих потоков.