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-214Б-24

Студент: Ельцова Д.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 30.10.25

Москва, 2024

Постановка задачи

Вариант 5.

Отсортировать массив целых чисел при помощи четно-нечетной сортировки Бетчера.

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.

Общий метод и алгоритм решения

В данной лабораторной работе использован POSIX Threads для создания многопоточной программы.

Основные использованные системные вызовы и функции:

- pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void*), void *arg); – создаёт новый поток, который начинает выполнение функции start_routine с аргументом arg.
- pthread_join(pthread_t thread, void **retval); – ожидает завершения указанного потока и возвращает его результат.
- malloc(size_t size); – выделение памяти для массивов.
- free(void *ptr); – освобождение выделенной памяти.
- gettimeofday(struct timeval *tv, struct timezone *tz); – получение текущего времени с точностью до микросекунд.
- memcpy(void *dest, const void *src, size_t n); – копирование данных между массивами.
- rand(); – генерация случайных чисел для заполнения массива.

Описание программы

Входные данные: Программа принимает 2 аргумента командной строки – размер массива и количество потоков.

Основные шаги работы программы:

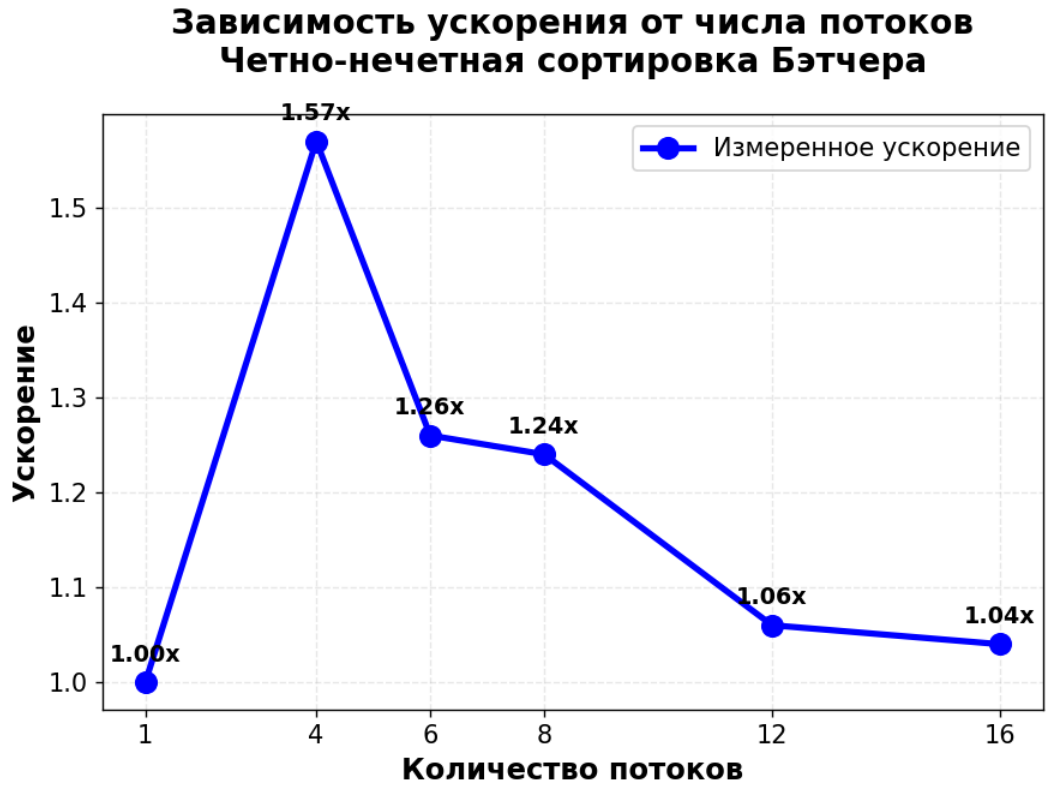
1. Генерация случайного массива. Используется функция generate_array(), которая заполняет массив случайными числами от 0 до 9999.
2. Последовательная сортировка Бетчера. Вызывается sequential_batcher_sort(), которой передаются сам массив целых чисел и его размер.
   Эта функция реализует реализует классический алгоритм четно-нечетной сортировки:
   - Алгоритм проходит через массив в несколько фаз.
   - На четных фазах сравниваются элементы с индексами (0,1), (2,3), (4,5)...
   - На нечетных фазах сравниваются элементы с индексами (1,2), (3,4), (5,6)...
   - Процесс повторяется до тех пор, пока на всей фазе не будет произведено ни одного обмена.
3. Параллельная сортировка Бетчера. Вызывается parallel_batcher_sort().
   Алгоритм:
   - Делим массив на P сегментов (где P = количество потоков)
   - Каждый поток независимо сортирует свой блок с помощью sequential_batcher_sort().

- После сортировки блоков выполняется финальная глобальная четно-нечетная сортировка для объединения результатов.

4. Сравнение времени и вычисление эффективности. Последовательное и параллельное выполнение замеряется с помощью функции get_current_time() (использует gettimeofday). Ускорение: speedup = t_seq / t_par; эффективность: eff = speedup / num_threads. Вывод результата в консоль.

Анализ ускорения и эффективности

Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	8387.03	1	1
4	5710.68	1.57	0.39
6	7145.48	1.26	0.21
8	7365.07	1.24	0.15
12	7719.27	1.06	0.09
16	7655.63	1.04	0.07
1024	7568.58	1.20	0.0012



Из графика четко видна оптимальная зона параллелизации при 4 потоках, где достигается максимальное ускорение 1.57x. При дальнейшем увеличении числа потоков наблюдается спад производительности - ускорение снижается до 1.04x при 16 потоках. Это характерное поведение для алгоритмов с высокой вычислительной сложностью, где накладные расходы на синхронизацию начинают преобладать над выигрышем от параллелизации.

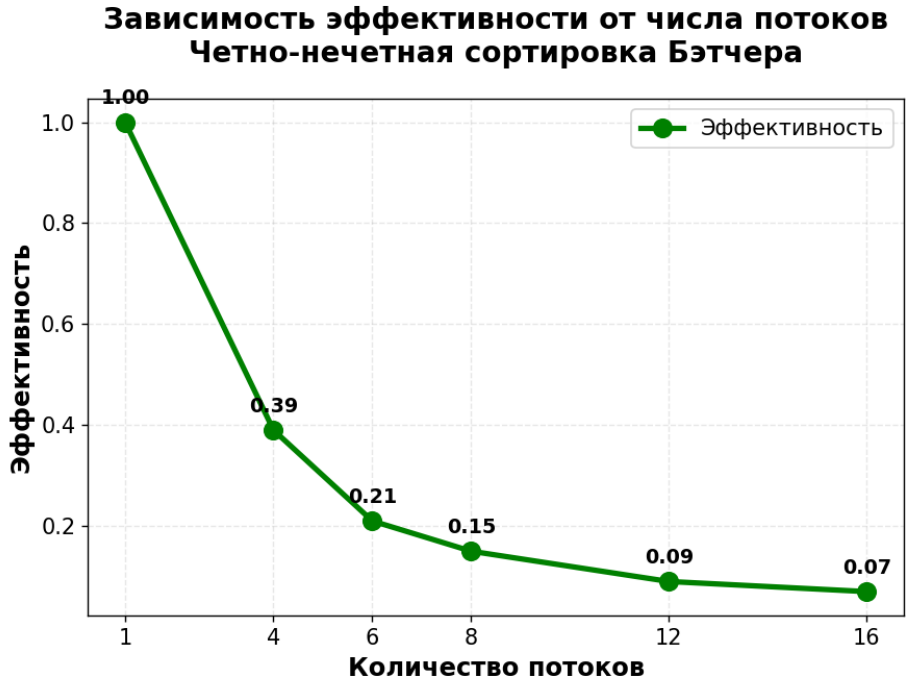


График эффективности демонстрирует резкое падение после 4 потоков - с 39% до 7% при 16 потоках. Наиболее эффективное использование ресурсов системы наблюдается именно при 4 потоках, где каждый поток работает с эффективностью 39%. Дальнейшее увеличение числа потоков приводит к нерациональному использованию вычислительных ресурсов.

Данные графики наглядно иллюстрируют закон Амдала, который ограничивает рост производительности при увеличении числа вычислителей. Для алгоритма четно-нечетной сортировки Бэтчера с его $O(n^2)$ сложностью и необходимостью финальной синхронизации, параллельный потенциал существенно ограничен последовательной частью алгоритма.

Для данного алгоритма оптимальным является использование 4 потоков, где достигается баланс между ускорением и эффективностью использования ресурсов. Дальнейшее увеличение числа потоков не только не дает существенного выигрыша в производительности, но и приводит к резкому падению эффективности, что делает такой подход нерациональным с точки зрения энергопотребления и загрузки системы.

Код программы

sorting.h

```
#ifndef SORTING_H
#define SORTING_H

#include <stddef.h>

double get_current_time(void);
void generate_array(int *array, int size);
int is_sorted(int *array, int size);
void sequential_batcher_sort(int arr[], int n);

// Параллельная версия (использует pthread_create)
void parallel_batcher_sort(int *array, int n, int num_threads);

#endif
```

sorting.c

```
#include "sorting.h"
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <sys/time.h>
#include <unistd.h>
#include <stdio.h>

//  Вспомогательные функции
double get_current_time(void) {
    struct timeval tv;
    gettimeofday(&tv, NULL);
```

```
    return tv.tv_sec + tv.tv_usec / 1000000.0;
}

void generate_array(int *array, int size) {
    for (int i = 0; i < size; ++i) array[i] = rand() % 10000;
}

int is_sorted(int *array, int size) {
    for (int i = 0; i + 1 < size; ++i) {
        if (array[i] > array[i + 1]) return 0;
    }
    return 1;
}

// ======================== Последовательная четно-нечетная сортировка
========================

void sequential_batcher_sort(int arr[], int n) {
    if (n <= 1) return;
    int sorted;
    do {
        sorted = 1;
        for (int i = 0; i < n - 1; i += 2) {
            if (arr[i] > arr[i + 1]) {
                int tmp = arr[i]; arr[i] = arr[i + 1]; arr[i + 1] = tmp;
                sorted = 0;
            }
        }
        for (int i = 1; i < n - 1; i += 2) {
            if (arr[i] > arr[i + 1]) {
                int tmp = arr[i]; arr[i] = arr[i + 1]; arr[i + 1] = tmp;
                sorted = 0;
            }
```

```
        }
    } while (!sorted);
}

// ======================== Параллельная версия (ГАРАНТИРОВАННО КОРРЕКТНАЯ)
========================

typedef struct {
    int *array;
    int start;
    int len;
} BlockArgs;

static void *sort_block_thread(void *arg) {
    BlockArgs *args = (BlockArgs*)arg;
    sequential_batcher_sort(args->array + args->start, args->len);
    return NULL;
}

void parallel_batcher_sort(int *array, int n, int num_threads) {
    if (n <= 1 || num_threads <= 1) {
        sequential_batcher_sort(array, n);
        return;
    }

    int P = num_threads;
    if (P > n/2) P = n/2;

    // 1. Параллельная сортировка блоков
    pthread_t threads[P];
    BlockArgs args[P];

    int block_size = n / P;
```

```
    for (int i = 0; i < P; i++) {
        args[i].array = array;
        args[i].start = i * block_size;
        args[i].len = (i == P - 1) ? (n - i * block_size) : block_size;
        pthread_create(&threads[i], NULL, sort_block_thread, &args[i]);
    }

    // Ждем завершения всех потоков
    for (int i = 0; i < P; i++) {
        pthread_join(threads[i], NULL);
    }

    // 2. Финальная последовательная сортировка (гарантирует корректность)
    sequential_batcher_sort(array, n);
}
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "sorting.h"

static void write_exit(const char *msg) {
    size_t len = strlen(msg);
    ssize_t result = write(STDOUT_FILENO, msg, len);
    (void)result;
    exit(EXIT_FAILURE);
}

int main(int argc, char **argv) {
    if (argc < 3) {
```

```
    write_exit("Использование: ./lab2 <размер_массива> <количество_потоков>\n");
}

int array_size = atoi(argv[1]);
int num_threads = atoi(argv[2]);

if (array_size <= 0) {
    write_exit("Ошибка: размер массива должен быть положительным числом\n");
}
if (num_threads <= 0) {
    write_exit("Ошибка: количество потоков должно быть положительным числом\n");
}

printf("PID: %d\n", getpid()); // полезно при демонстрации потоков ОС

int *original = malloc(array_size * sizeof(int));
int *seq_result = malloc(array_size * sizeof(int));
int *par_result = malloc(array_size * sizeof(int));

if (!original || !seq_result || !par_result) {
    write_exit("Ошибка выделения памяти\n");
}

srand((unsigned)time(NULL));
generate_array(original, array_size);
memcpy(seq_result, original, array_size * sizeof(int));
memcpy(par_result, original, array_size * sizeof(int));

// Последовательная сортировка
double seq_t0 = get_current_time();
sequential_batcher_sort(seq_result, array_size);
double seq_time = get_current_time() - seq_t0;
```

```
    // Параллельная сортировка (замер полной параллельной процедуры)
    double par_t0 = get_current_time();
    parallel_batcher_sort(par_result, array_size, num_threads);
    double par_time = get_current_time() - par_t0;

    double speedup = seq_time / par_time;
    double eff = speedup / num_threads;

    int seq_correct = is_sorted(seq_result, array_size);
    int par_correct = is_sorted(par_result, array_size);

    char output[512];
    int len = 0;

    len += snprintf(output + len, sizeof(output) - len,
                    "Последовательная сортировка завершена за: %.2f мс\n", seq_time *
1000.0);
    len += snprintf(output + len, sizeof(output) - len,
                    "Параллельная сортировка завершена за: %.2f мс\n", par_time *
1000.0);
    len += snprintf(output + len, sizeof(output) - len,
                    "Ускорение: %.2fx\n", speedup);
    len += snprintf(output + len, sizeof(output) - len,
                    "Эффективность: %.2f\n", eff);

    if (seq_correct && par_correct) {
        len += snprintf(output + len, sizeof(output) - len,
                        "Корректная сортировка: ДА\n");
    } else {
        len += snprintf(output + len, sizeof(output) - len,
                        "Корректная сортировка: НЕТ\n");
    }

    write(STDOUT_FILENO, output, len);
```

```
    free(original);

    free(seq_result);

    free(par_result);


    return 0;

}
```

Протокол работы программы

Тестирование:

```
suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr2$ gcc -O0 -pthread sorting.c main.c -o lab2

suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr2$ ./lab2 100000 4

PID: 308955

Размер массива: 100000, Потоки: 4

Последовательная: 8988.68 мс, Корректность: ДА

Параллельная: 5710.68 мс, Корректность: ДА

Ускорение: 1.57x

Эффективность: 0.39

Результаты идентичны: ДА

suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr2$ ./lab2 100000 6

PID: 308960

Размер массива: 100000, Потоки: 6

Последовательная: 9009.79 мс, Корректность: ДА

Параллельная: 7145.48 мс, Корректность: ДА

Ускорение: 1.26x

Эффективность: 0.21

Результаты идентичны: ДА

suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr2$ ./lab2 100000 8

PID: 308967

Размер массива: 100000, Потоки: 8

Последовательная: 9109.55 мс, Корректность: ДА

Параллельная: 7365.07 мс, Корректность: ДА
```

Ускорение: 1.24x

Эффективность: 0.15

Результаты идентичны: ДА

suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr2$ ./lab2 100000 12

PID: 308976

Размер массива: 100000, Потоки: 12

Последовательная: 8200.03 мс, Корректность: ДА

Параллельная: 7719.27 мс, Корректность: ДА

Ускорение: 1.06x

Эффективность: 0.09

Результаты идентичны: ДА

suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr2$ ./lab2 100000 16

PID: 308989

Размер массива: 100000, Потоки: 16

Последовательная: 7971.88 мс, Корректность: ДА

Параллельная: 7655.63 мс, Корректность: ДА

Ускорение: 1.04x

Эффективность: 0.07

Результаты идентичны: ДА

suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr2$ ./lab2 100000 1024

PID: 309006

Размер массива: 100000, Потоки: 1024

Последовательная: 9116.34 мс, Корректность: ДА

Параллельная: 7568.58 мс, Корректность: ДА

Ускорение: 1.20x

Эффективность: 0.00

Результаты идентичны: ДА

suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr2$ ./lab2 200000 4

PID: 310031

Размер массива: 200000, Потоки: 4

Последовательная: 42617.35 мс, Корректность: ДА

Параллельная: 28322.65 мс, Корректность: ДА

Ускорение: 1.50x

Эффективность: 0.38

Результаты идентичны: ДА

Strace:

```
suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr2$ strace -f ./lab2 100000 4

execve("./lab2", ["./lab2", "100000", "4"], 0x7ffdf052a9a8 /* 28 vars */) = 0

brk(NULL)                               = 0x555b9d754000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f3d72e82000

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

fstat(3, {st_mode=S_IFREG|0644, st_size=19963, ...}) = 0

mmap(NULL, 19963, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f3d72e7d000

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\0\220\243\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

fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 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, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f3d72c6b000

mmap(0x7f3d72c93000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f3d72c93000

mmap(0x7f3d72e1b000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f3d72e1b000

mmap(0x7f3d72e6a000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f3d72e6a000

mmap(0x7f3d72e70000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f3d72e70000

close(3)                                = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f3d72c68000

arch_prctl(ARCH_SET_FS, 0x7f3d72c68740) = 0

set_tid_address(0x7f3d72c68a10)         = 310154

set_robust_list(0x7f3d72c68a20, 24)     = 0

rseq(0x7f3d72c69060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f3d72e6a000, 16384, PROT_READ) = 0
```

```
mprotect(0x555b79362000, 4096, PROT_READ) = 0

mprotect(0x7f3d72eba000, 8192, PROT_READ) = 0

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

munmap(0x7f3d72e7d000, 19963)           = 0

getpid()                                = 310154

fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0

getrandom("\x20\xe3\xa5\x55\x92\x64\x32\x5c", 8, GRND_NONBLOCK) = 8

brk(NULL)                               = 0x555b9d754000

brk(0x555b9d775000)                     = 0x555b9d775000

write(1, "PID: 310154\n", 12PID: 310154

)           = 12

mmap(NULL, 401408, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f3d72c06000

mmap(NULL, 401408, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f3d72ba4000

mmap(NULL, 401408, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f3d72b42000

rt_sigaction(SIGRT_1, {sa_handler=0x7f3d72d04530, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f3d72cb0330}, 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) =
0x7f3d72341000

mprotect(0x7f3d72342000, 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_SY
SVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f3d72b41990, parent_tid=0x7f3d72b41990, exit_signal=0,
stack=0x7f3d72341000, stack_size=0x7fff80, tls=0x7f3d72b416c0}strace: Process 310157
attached

 => {parent_tid=[310157]}, 88) = 310157

[pid 310154] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>

[pid 310157] rseq(0x7f3d72b41fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 310154] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 310154] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0
<unfinished ...>

[pid 310157] <... rseq resumed>)        = 0

[pid 310154] <... mmap resumed>)        = 0x7f3d71b40000

[pid 310157] set_robust_list(0x7f3d72b419a0, 24 <unfinished ...>

[pid 310154] mprotect(0x7f3d71b41000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>
```

```
[pid 310157] <... set_robust_list resumed>) = 0

[pid 310154] <... mprotect resumed>)    = 0

[pid 310157] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>

[pid 310154] rt_sigprocmask(SIG_BLOCK, ~[],  <unfinished ...>

[pid 310157] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 310154] <... rt_sigprocmask resumed>[], 8) = 0

[pid 310154]
```
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f3d72340990, parent_tid=0x7f3d72340990, exit_signal=0, stack=0x7f3d71b40000, stack_size=0x7fff80, tls=0x7f3d723406c0}strace: Process 310158 attached

```
 => {parent_tid=[310158]}, 88) = 310158

[pid 310158] rseq(0x7f3d72340fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 310154] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>

[pid 310158] <... rseq resumed>)        = 0

[pid 310154] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 310158] set_robust_list(0x7f3d723409a0, 24 <unfinished ...>

[pid 310154] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0
<unfinished ...>

[pid 310158] <... set_robust_list resumed>) = 0

[pid 310154] <... mmap resumed>)        = 0x7f3d7133f000

[pid 310158] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>

[pid 310154] mprotect(0x7f3d71340000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>

[pid 310158] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 310154] <... mprotect resumed>)    = 0

[pid 310154] rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

[pid 310154]
```
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f3d71b3f990, parent_tid=0x7f3d71b3f990, exit_signal=0, stack=0x7f3d7133f000, stack_size=0x7fff80, tls=0x7f3d71b3f6c0}strace: Process 310159 attached

=> {parent_tid=[310159]}, 88) = 310159

```
[pid 310159] rseq(0x7f3d71b3ffe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 310154] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>

[pid 310159] <... rseq resumed>)        = 0

[pid 310154] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 310159] set_robust_list(0x7f3d71b3f9a0, 24 <unfinished ...>

[pid 310154] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0
<unfinished ...>

[pid 310159] <... set_robust_list resumed>) = 0
```

[pid 310154] <... mmap resumed>) = 0x7f3d70b3e000

[pid 310159] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 310154] mprotect(0x7f3d70b3f000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>

[pid 310159] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 310154] <... mprotect resumed>) = 0

[pid 310154] rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

[pid 310154]
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f3d7133e990, parent_tid=0x7f3d7133e990, exit_signal=0, stack=0x7f3d70b3e000, stack_size=0x7fff80, tls=0x7f3d7133e6c0}strace: Process 310160 attached

=> {parent_tid=[310160]}, 88) = 310160

[pid 310160] rseq(0x7f3d7133efe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 310154] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 310160] <... rseq resumed>) = 0

[pid 310154] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 310160] set_robust_list(0x7f3d7133e9a0, 24 <unfinished ...>

[pid 310154] futex(0x7f3d72b41990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 310157, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 310160] <... set_robust_list resumed>) = 0

[pid 310160] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

[pid 310157] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0

[pid 310157] madvise(0x7f3d72341000, 8368128, MADV_DONTNEED) = 0

[pid 310157] exit(0) = ?

[pid 310154] <... futex resumed>) = 0

[pid 310157] +++ exited with 0 +++

[pid 310154] futex(0x7f3d72340990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 310158, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 310159] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0

[pid 310159] madvise(0x7f3d7133f000, 8368128, MADV_DONTNEED) = 0

[pid 310159] exit(0) = ?

[pid 310159] +++ exited with 0 +++

[pid 310158] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0

[pid 310158] madvise(0x7f3d71b40000, 8368128, MADV_DONTNEED) = 0

[pid 310158] exit(0) = ?

[pid 310154] <... futex resumed>) = 0

[pid 310158] +++ exited with 0 +++

```
      [pid 310154] futex(0x7f3d7133e990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 310160,
NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

      [pid 310160] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0

      [pid 310160] madvise(0x7f3d70b3e000, 8368128, MADV_DONTNEED) = 0

      [pid 310160] exit(0)                    = ?

      [pid 310154] <... futex resumed>)       = 0

      [pid 310160] +++ exited with 0 +++

      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\260\321\217"..., 283Последовательная сортировка завершена за:
7852.75 мс

      Параллельная сортировка завершена за: 7023.58 мс

      Ускорение: 1.12x

      Эффективность: 0.28

      Корректная сортировка: ДА

      ) = 283

      munmap(0x7f3d72c06000, 401408)          = 0

      munmap(0x7f3d72ba4000, 401408)          = 0

      munmap(0x7f3d72b42000, 401408)          = 0

      exit_group(0)                           = ?

      +++ exited with 0 +++

      suslik@WIN-L3ULFBUQJMS:/mnt/c/Users/daria/Desktop/уч и зад/3 сем/оси/lr2$
```

Вывод

В ходе выполнения лабораторной работы были успешно изучены и применены основные системные вызовы POSIX Threads для создания многопоточной программы сортировки. Была реализована параллельная версия четно-нечетной сортировки Бэтчера, позволяющая оценить эффективность параллелизации вычислительных задач. Экспериментальные результаты показали, что максимальное ускорение 1.57x достигается при 4 потоках с эффективностью 39%. Несмотря на параллелизацию, общая сложность алгоритма остается $O(n^2)$ из-за доминирующей финальной последовательной стадии, что объясняет ограниченный потенциал ускорения согласно закону Амдала.