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-210БВ-24

Студент: Кудряшова Т.И.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 02.10.25

Москва, 2025

Постановка задачи

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

Условия правильного выполнения:

1. Не использовать стандартную библиотеку, только POSIX Threads (для Linux и macOS) и Windows API.
2. Реализовать две версии алгоритма по вашему варианту: последовательный и параллельный.
3. Замерить время выполнение работы для обеих версий, причём для параллельной версии нужно замерять время для каждого параметра количетсва потоков, которое вы выделяете.
4. Добавить в отчет таблицу в формате:

Число потоков	Время исполнения (мс)	Ускорение	Эффективность

   Помимо этого в таблице должно содержаться:
   - 4 эксперимента с количеством потоков меньше количества логических ядер на вашей системе
   - 1 эксперимент с количеством потоков равным количеству логических ядер на вашей системе
   - 3 эксперимента с количеством потоков сильно больше (например 16, 128, 1024) количества логических ядер на вашей системе
5. Добавление в отчёт метрик ускорения и эффективости (назначение, как вычислить и пояснение за переменные).
6. В выводе должно быть объясние полученных результатов в таблице и графике.

Вариант 5.

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

Общий метод и алгоритм решения.

Теоретическая база.

Сортировка Бетчера.

Основные принципы:

1. Четно-нечетное слияние (Odd-even merge).

Алгоритм использует операцию четно-нечетного слияния двух отсортированных последовательностей.

2. Сравнение-обмен (Compare-exchange)

Основная операция - сравнение двух элементов и их обмен, если они не в правильном порядке.

Алгоритм сортировки.

1. Получаем на вход массив целых чисел и их количество(n).
2. Запускаем цикл(p) по уровням сортировки: 1, 2, 4, 8 и т.д. до n.
3. На каждом уровне проходимся по расстояниям между элементами(k): p, p/2, p/4, … пока k >= 1.
4. Сравниваем элементы массива по парам на расстоянии k, если порядок неверен, меняем местами.
5. Возвращаем обновленный массив.

В своей работе используем упрощение этой сортировки – четная-нечетная сортировка Бетчера. В ней используется только 2 уровня сортировки с расстоянием в 2 элемента.

Алгоритм программы.

Реализуем 2 варианта сортировки – последовательную и непоследовательную. Выбор будем делать по полученному в командной строке количеству потоков.

Реализуем вспомогательную функцию для вычисления времени начала и конца работы функции. Последовательная сортировка – реализации четно-нечетной сортировки Бетчера в явном ее виде(см выше). Параллельная ее версия будет использовать разное количество потоков для сортировки отдельных пар чисел используя id как отступ, а общее количество – разделение работы потоков. Каждый поток на 1 шаг из num_threads возьмет id пару чисел.

Системные вызовы:

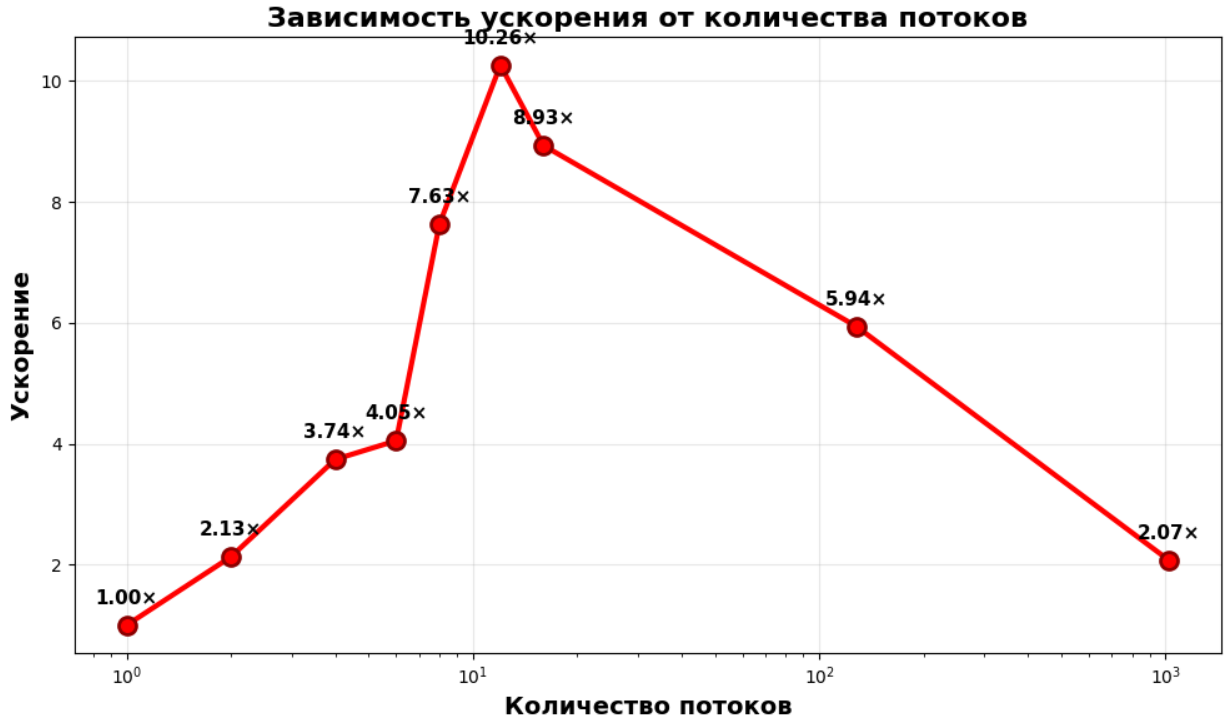
- malloc(sizeof(int) * n) – выделение динамической памяти
- pthread_create(&threads[i], NULL, thread_func, &data[i]) – создание нового потока выполнения
- pthread_join(threads[i], NULL) – блокировка до завершения указанного потока
- pthread_barrier_init(&barrier, NULL, num_threads) – инициализация барьерной синхронизации
- pthread_barrier_wait(&barrier) – ожидание достижения барьера
- pthread_barrier_destroy(&barrier) – освобождение ресурсов барьера
- free(array) – освобождение памяти

Получение метрик:

Количество ядер – 6.

Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	203.140	1	1
2	95.321	2.13	1.07
4	54.280	3.74	0.94
8	26.612	7.63	0.95
6	50.196	4.05	0.68
12	19.803	10.26	0.85
16	22.756	8.93	0.56
128	34.201	5.94	0.05
1024	98.358	2.07	0.02

Графики:



Код программы.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include "functions.c"

int *array;
int n;
int num_threads;
pthread_barrier_t barrier;

void odd_even_sort_seq(int *arr, int n) {
    int sorted = 0;
    while (!sorted) {
        sorted = 1;
        // Четная фаза
        for (int i = 0; i < n - 1; i += 2) {
            if (arr[i] > arr[i + 1]) {
                swap(&arr[i], &arr[i + 1]);
                sorted = 0;
            }
        }
        // Нечетная фаза
        for (int i = 1; i < n - 1; i += 2) {
            if (arr[i] > arr[i + 1]) {
                swap(&arr[i], &arr[i + 1]);
                sorted = 0;
            }
        }
    }
}

pthread_barrier_t barrier;
```

```c
// структура потока
typedef struct {
    int id;
} thread_data_t;

void* thread_func(void* arg) {
    thread_data_t *data = (thread_data_t*)arg;
    int id = data->id;

    int phase, i, start;
    for (phase = 0; phase < n; phase++) {
        start = (phase % 2 == 0) ? 0 : 1;

        for (i = start + id * 2; i < n - 1; i += num_threads * 2) {
            if (array[i] > array[i + 1])
                swap(&array[i], &array[i + 1]);
        }
    }
    pthread_barrier_wait(&barrier);
    return NULL;
}

void odd_even_sort_parallel(int *arr, int n, int num_threads) {
    pthread_t threads[num_threads];
    thread_data_t data[num_threads];

    pthread_barrier_init(&barrier, NULL, num_threads);

    for (int i = 0; i < num_threads; i++) {
        data[i].id = i;
        pthread_create(&threads[i], NULL, thread_func, &data[i]);
    }

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

    pthread_barrier_destroy(&barrier);
}

int main(int argc, char *argv[]) {
    if (argc < 3) {
        printf("You had to enter array_size and num_threads\n");
        return 1;
    }

    char *endptr1, *endptr2;
    n = (int)strtol(argv[1], &endptr1, 10);
    num_threads = (int)strtol(argv[2], &endptr2, 10);
    if (*endptr1 != '\0' || *endptr2 != '\0' || num_threads <= 0 || n <= 0) {
        printf("Invalid input data: %s %s\n", argv[1], argv[2]);
        return 1;
    }

    array = malloc(sizeof(int) * n);

    // printf("Enter %d integers:\n", n);
    // for (int i = 0; i < n; i++) {
    //     scanf("%d", &array[i]);
    // }
    printf("Generating array of size %d...\n", n);
    for (int i = 0; i < n; i++) {
        array[i] = rand() % 10000;
    }
```

```
    double start_time, end_time, elapsed_time;

    if (num_threads == 1) {
        start_time = get_time_ms();
        odd_even_sort_seq(array, n);
        end_time = get_time_ms();
        elapsed_time = end_time - start_time;

        printf("Linear time: %.3f\n", elapsed_time);
        // printArray(array, n);
    } else {
        start_time = get_time_ms();
        odd_even_sort_parallel(array, n, num_threads);
        end_time = get_time_ms();
        elapsed_time = end_time - start_time;

        printf("Paralel time (%d threads): %.3f\n", num_threads, elapsed_time);
        // printArray(array, n);

        // Проверка корректности
        char mistake[256] = "";
        int result = check_sort_correctness(array, n, mistake);
        if (!result) {
            printf("%s\n", mistake);
        }
    }
    free(array);
    return 0;
}
```

Протокол работы программы.

./program 10000 1

Linear time: 203.140

./program 10000 2

Paralel time (2 threads): 95.321

./program 10000 8

Paralel time (8 threads): 26.612

Вывод.

В теории программа проста в выполнении, но хорошая эффективность будет зависеть от многих факторов. Главный из которых – понимание программиста сути происходящего. Также для меня в итоге осталась неясна аномалия в районе 6 ядер. Вроде в теории это должно быть наиболее эффективно, но из-за, видимо, оптимизации ОС на 8, 12 и тд эффективность алгоритма при 6 потоках откровенно удручает.