

Московский Авиационный Институт  
(Национальный Исследовательский Университет)  
Факультет информационных технологий и прикладной математики  
Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**  
**«Операционные системы»**

Студент: Знай Артемий Олегович  
Группа: М80-201Б-21  
Вариант: 3  
Преподаватель: Миронов Евгений Сергеевич  
Оценка: \_\_\_\_\_  
Дата: \_\_\_\_\_  
Подпись: \_\_\_\_\_

Москва, 2022

## **Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

## Репозиторий

[https://github.com/znako/OS\\_LABS/](https://github.com/znako/OS_LABS/)

## Постановка задачи

### Цель работы

Целью является приобретение практических навыков в:

- Управление потоками в ОС
- Обеспечение синхронизации между потоками

### Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков 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 входящих данных и количества потоков. Получившиеся результаты необходимо объяснить.

### Вариант 3

Отсортировать массив целых чисел при помощи параллельной сортировки слиянием

### Общие сведения о программе

Программа компилируется из файла main.cpp. Также используются заголовочные файлы: lab3.h, iostream, stdio.h, stdlib.h, vector, pthread.h. В программе используются следующие системные вызовы:

1. pthread\_create() – создание нового потока
2. pthread\_join() – ожидание окончания потока

### Общий метод и алгоритм решения

На вход подается длина вектора целых чисел, количество потоков, которые его будут обрабатывать и сам вектор целых чисел. Он разделяется на количество потоков, создаются потоки, которые сортируют каждый свой кусок в функции потока, далее эти куски сливаются в цельный отсортированный массив.

### Исходный код

#### Main.cpp

```
#include "lab3.h"

#include <iostream>

int main() {
    int length, threadCount;
```

```

std::cin >> length >> threadCount;
std::vector<int> arr(length);

for (int i = 0; i < length; i++) {
    std::cin >> arr[i];
}

MergeSortArr(arr, threadCount);

std::cout << "result ";
for (int i = 0; i < length; i++) {
    std::cout << arr[i] << ' ';
}
std::cout << '\n';
}

```

### Lab3.cpp

```

#include "lab3.h"
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <pthread.h>
#include <vector>

struct TArg {
    std::vector<int> &a;
    long int i;
    int numbersPerThread;
    int offset;
    int threadCount;
};

// Главная функция управления потоками и сортировкой
void MergeSortArr(std::vector<int> &array, int threadCount) {

    int numbersPerThread = array.size() / threadCount;
    int offset = array.size() % threadCount;

    if (threadCount > 1) {
        std::vector<pthread_t> threads(threadCount);
        std::vector<TArg> arglist;
        arglist.reserve(threadCount);

        for (long i = 0; i < threadCount; i++) {

            arglist.push_back({array, i, numbersPerThread, offset, threadCount});
            int rc = pthread_create(&threads[i], NULL, ThreadMergeSort, &arglist[i]);
            if (rc) {
                std::cerr << "Thread create error";
            }
        }

        for(long i = 0; i < threadCount; i++) {
            pthread_join(threads[i], NULL);
        }
        int low = numbersPerThread;
        int high;
        for (int i = 1; i < threadCount; i++) {
            high = low + numbersPerThread - 1;
            if (i == (threadCount - 1)) {
                high = array.size() - 1;
            }
            Merge(array, 0, low - 1, high);
            low += numbersPerThread;
        }
    }
}

```

```

    }
}
else {
    MergeSort(array, 0, array.size()-1);
}
}

// Функция потока
void *ThreadMergeSort(void* arg) {
    TArg* argum = (TArg*)arg;
    int thread_id = (long)argum->i;
    int left = thread_id * (argum->numbersPerThread);
    int right = (thread_id + 1) * (argum->numbersPerThread) - 1;
    if (thread_id == argum->threadCount - 1) {
        right += argum->offset;
    }
    int middle = left + (right - left) / 2;
    if (left < right) {

        MergeSort(argum->a, left, right);
        MergeSort(argum->a, left + 1, right);
        Merge(argum->a, left, middle, right);

    }
    return NULL;
}

// merge sort
void MergeSort(std::vector<int> &arr, int left, int right) {
    if (left < right) {
        int middle = left + (right - left) / 2;
        MergeSort(arr, left, middle);
        MergeSort(arr, middle + 1, right);
        Merge(arr, left, middle, right);
    }
}

void Merge(std::vector<int> &arr, int left, int middle, int right) {
    int i = 0;
    int j = 0;
    int k = 0;
    int left_length = middle - left + 1;
    int right_length = right - middle;
    std::vector<int> left_array(left_length);
    std::vector<int> right_array(right_length);

    // копируем значения в левый массив
    for (int i = 0; i < left_length; i++) {
        left_array[i] = arr[left + i];
    }

    // Копируем значения в правый массив
    for (int j = 0; j < right_length; j++) {
        right_array[j] = arr[middle + 1 + j];
    }

    i = 0;
    j = 0;
    // Выбираем значения из левого и правого массивов и заносим
    while (i < left_length && j < right_length) {
        if (left_array[i] <= right_array[j]) {
            arr[left + k] = left_array[i];
            i++;

```

```

        } else {
            arr[left + k] = right_array[j];
            j ++;
        }
        k ++;
    }

    // Заносим оставшиеся значения
    while (i < left_length) {
        arr[left + k] = left_array[i];
        k ++;
        i ++;
    }
    while (j < right_length) {
        arr[left + k] = right_array[j];
        k ++;
        j ++;
    }
}

```

## Lab3.h

```

#ifndef OS_LABS_LAB3_H
#define OS_LABS_LAB3_H

#include <vector>

void MergeSortArr(std::vector<int> &arr, int threadCount);
void MergeSort(std::vector<int> &arr, int left, int right);
void Merge(std::vector<int> &arr, int left, int middle, int right);
void* ThreadMergeSort(void* arg);

#endif //OS_LABS_LAB3_H

```

## Демонстрация работы программы

```
znako@znako-VirtualBox:~/UtoW/OS_LABS/build/lab3$ ./lab3
```

```
6 2
```

```
23241
```

```
124
```

```
123
```

```
46546
```

```
575
```

```
57
```

```
result 57 123 124 575 23241 46546
```

```
znako@znako-VirtualBox:~/UtoW/OS_LABS/build/lab3$ ./lab3
```

```
10 1
```

```
56789
```

```
124
```

```
1361
```

```
12
```

```
53
```

```
6
```

```
769
```

```
2
```

```
578
```

```
2
```

```
result 2 2 6 12 53 124 578 769 1361 56789
```

## Выводы

Составлена и отлажена многопоточная программа на языке C++, выполняющая параллельную сортировку слиянием. Тем самым, приобретены навыки в распараллеливании вычислений, управлении потоками и обеспечении синхронизации между ними.