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: Марченко Алексей Эдуардович

Группа: М8О-207Б-21

Вариант: 5

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

**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/MarchAleksey/OS_Labs>

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в управлении процессами 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 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 5**

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

**Общие сведения о программе**

Программа компилируется из файла main.c

**Общий метод и алгоритм решения**

Код начинается с включения необходимых заголовочных файлов и определения переменной типа pthread\_mutex\_t для создания мьютекса.

Затем определяются две структуры данных - data\_ и data. Структура data\_ определяет поля, необходимые для одного шага сортировки массива, а структура data объединяет эти поля вместе.

Далее определяется функция thread\_func, которая представляет функцию, которую будут выполнять потоки. Функция получает на вход указатель на структуру data и выполняет сортировку элементов массива, индексы которых находятся между ind\_first и ind\_last. При этом используется алгоритм сортировки Шелла.

Затем определяется функция main, которая является точкой входа в программу. В начале функция проверяет количество переданных аргументов командной строки и если их количество больше двух, то выводит сообщение об использовании и завершает программу с кодом 1. Если передан аргумент, то он используется как количество потоков, которые будут создаваться в процессе работы программы. Если аргумент не передан, то количество потоков равно 1.

Потом программа считывает размер массива N с помощью scanf и вычисляет ближайшую степень двойки, которая не меньше N. Затем программа считывает элементы массива и заполняет его значениями. Если N не является степенью двойки, то оставшиеся элементы массива заполняются максимальным значением типа int.

Следом программа проверяет, что количество потоков не больше N/2, если это так, то количество потоков устанавливается на максимально возможное значение, т.е. степень двойки, которая наиболее близка к N/2.

Далее программа начинает выполнять сортировку. Для этого используется цикл, который работает пока шаг d сортировки не станет равным 1. На каждом шаге цикла программа создает потоки и выполняет сортировку с помощью функции thread\_func. Далее потоки ждут завершения своей работы с помощью функции pthread\_join и переходят на следующий шаг.

После завершения сортировки программа копирует отсортированные элементы массива в массив res и выводит его на экран.

**Исходный код**

**main.c**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#include <stdbool.h>

#include <errno.h>

#include <pthread.h>

#include <time.h>

pthread\_mutex\_t mutex;

typedef struct data\_{

    int ind\_first;

    int ind\_last;

    int d;

    int r;

    int p;

    int N;

    int step;

    int \*Arr;

} data\_;

typedef struct data\_ data;

void \*thread\_func(void \*arg) {

    pthread\_mutex\_lock(&mutex);

    data \*a = (data\*)arg;

    for (int j = a->ind\_first; j < a->ind\_last; j++) {

        if ((j & a->p) == a->r) {

            if (a->Arr[j] > a->Arr[j + a->d]){

                int temp = a->Arr[j];

                a->Arr[j] = a->Arr[j + a->d];

                a->Arr[j + a->d] = temp;

            }

        }

    }

    pthread\_mutex\_unlock(&mutex);

    return NULL;

}

int main(int argc, char\* argv[]) {

    int stm = 1;

    if (argc > 2) {

        perror("Usage: ./a.out <number of threads>\n");

        return 1;

    }

    else if (argc == 2) {

        stm = atoi(argv[1]);

    }

    int threads = stm;

    if (threads < 1) {

        perror("Incorrect number of threads\n");

        return 2;

    }

    int N;

    scanf("%d", &N);

    int st2 = 1;

    while (st2 < N) {

        st2 \*= 2;

    }

    int Arr[st2];

    for (int i = 0; i < N; i++) {

        scanf("%d", &Arr[i]);

    }

    for (int i = N; i < st2; i++) {

        Arr[i] = INT\_MAX;

    }

    if (stm >= st2 / 2) {

        stm = st2 / 2;

        threads = stm;

    }

    printf("Number of used threads: %d\n", threads);

    for (int p = st2; p > 0; p /= 2) {

        int q = st2, r = 0, d = p;

        bool b;

        pthread\_t thread[threads];

        do{

            int nTo = st2 - d;

            int k = nTo / threads;

            if (k < 2) {

                k = 2;

            }

            if (threads \* k < nTo) {

                k++;

            }

            int pt = 0;

            for(int i = 0; i < nTo; i += k) {

                data\* arg = malloc(sizeof(data));

                arg->ind\_first = i;

                if (i + k < nTo) {

                    arg->ind\_last = i + k;

                } else {

                    arg->ind\_last = nTo;

                }

                arg->d = d;

                arg->N = st2;

                arg->p = p;

                arg->r = r;

                arg->Arr = Arr;

                //printf("First %d Second %d\n", arg->ind\_first, arg->ind\_last);

                if (pthread\_create(&thread[pt], NULL, thread\_func, (void \*)arg) != 0) {

                    perror("Error: can't create thread");

                    return 3;

                }

                pt++;

            }

            for (int i = 0; i < pt; i++) {

                if (pthread\_join(thread[i], NULL) != 0) {

                    perror("Error: can't join thread");

                    return -3;

                }

            }

            b = q != p;

            if (b) {

                d = q - p;

                q >>= 1;

                r = p;

            }

        } while (b);

    }

    int res[N];

    for (int i = 0; i < N; i++) {

        res[i] = Arr[i];

    }

    printf("Sorted massive: ");

    for (int i = 0; i < N; i++) {

        printf("%d ", res[i]);

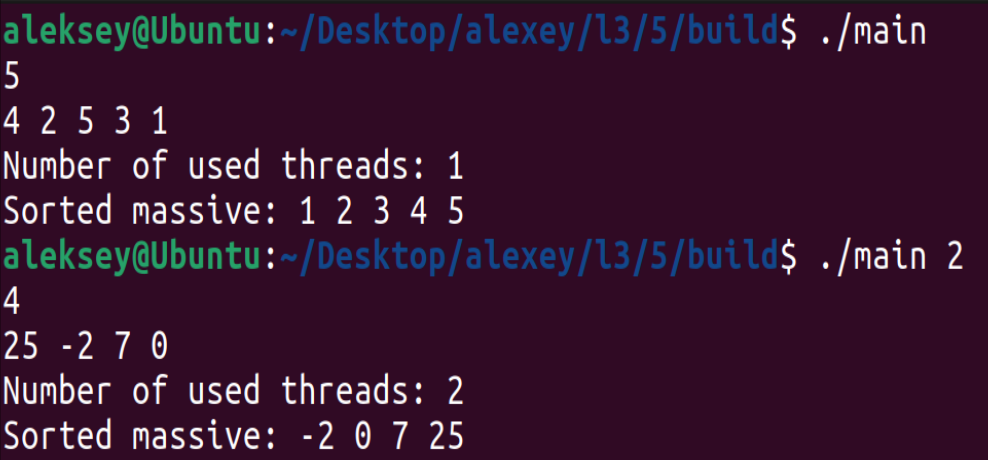
    }

    printf("\n");

    return 0;

}

**Демонстрация работы программы**

****

**Выводы**

В ходе выполнения лабораторной работы №3 я приобрел практические навыки в управлении процессами в ОС,и в обеспечении синхронизации между потоками.