**Московский авиационный институт (национальный исследовательский университет)**

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

**Лабораторная работа по предмету "Операционные системы" №2**

Студент: Федоров М. А.

Преподаватель: Миронов Е.С.

Группа: М8О-207Б-22

Дата: 15.09.2022

Оценка:

Подпись:

Оглавление

[Цель работы 3](#__RefHeading__738_2004454639)

[Постановка задачи 3](#__RefHeading__740_2004454639)

[Общие сведения о программе 4](#__RefHeading__742_2004454639)

[Общий алгоритм решения 5](#__RefHeading__744_2004454639)

[Реализация 5](#__RefHeading__746_2004454639)

[Пример работы 8](#__RefHeading__748_2004454639)

[Вывод 9](#__RefHeading__750_2004454639)

# **Цель работы**

Приобретение практических навыков в:

* Управление потоками в ОС
* Обеспечение синхронизации между потоками

# **Постановка задачи**

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.

*Вариант 5:*

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

# **Общие сведения о программе**

Программа представлена файлом **–** main2.c.

В программе используются следующие системные вызовы:

**fopen() –** открывает файл и возвращает его дескриптор.

**fscanf() –** считывает данные из файлового потока ввода.

**Pthread\_create()** **–** создание дочернего потока.

**Pthread\_join()** **–** ожидание сигнала от дочернего потока

**clock() –** получение текущего значения времени в системе.

# **Общий алгоритм решения**

# **Реализация**

**main2.c**

#include<stdio.h>

#include<pthread.h>

#include<stdlib.h>

#include<time.h>

void\* streamFunc();

int recursionSort();

void mergeSort();

int swapFunc();

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

    time\_t start = clock();

    time\_t end;

    if (argc != 2){

        printf("error: wrong number of arguments\n");

        return -1;

    }

    FILE \* readmas = fopen("mas", "r");

    int streams = strtol(argv[1], NULL, 10);

    int actual\_len;

    int formal\_len = 1;

    int for\_counter;

    int inheritable\_streams\_one = streams/2;

    int inheritable\_streams\_two = streams-inheritable\_streams\_one;

    int len\_1 = 0;

    int len\_2 = 0;

    pthread\_t stream\_one;

    pthread\_t stream\_two;

    fscanf(readmas, "%d", &actual\_len);

    for (formal\_len; formal\_len < actual\_len; formal\_len \*= 2);

    int mas[formal\_len];

    int returned\_mas\_1[formal\_len/2];

    int returned\_mas\_2[formal\_len/2];

    int\* args\_one[] = {&len\_1, returned\_mas\_1, &inheritable\_streams\_one};

    int\* args\_two[] = {&len\_2, returned\_mas\_2, &inheritable\_streams\_two};

    for (for\_counter = 0; for\_counter < actual\_len; ++for\_counter){

        fscanf(readmas, "%d", &mas[for\_counter]);

        printf("%d ", mas[for\_counter]);

    }

    for (for\_counter; for\_counter < formal\_len; ++for\_counter){

        mas[for\_counter] = -1;

        printf("%d ", mas[for\_counter]);

    }

    printf("\n\n");

    printf("streams:%d\n", streams);

    if (streams > 1) {

        for (int index = 1; index < formal\_len; index += 2){

            returned\_mas\_1[len\_1] = mas[index-1];

            returned\_mas\_2[len\_2] = mas[index];

            ++len\_1;

            ++len\_2;

        }

        pthread\_create(&stream\_one, NULL, streamFunc, args\_one);

        pthread\_create(&stream\_two, NULL, streamFunc, args\_two);

        pthread\_join(stream\_two, NULL);

        pthread\_join(stream\_one, NULL);

        mergeSort(len\_1, len\_2, returned\_mas\_1, returned\_mas\_2, mas);

    } else {

        recursionSort(formal\_len, mas);

    }

    printf("\nFirst 'sorted' 20 elements:\n");

    for (for\_counter = 0; for\_counter < formal\_len && for\_counter < 20; ++for\_counter){

        printf("%d ", mas[for\_counter]);

    }

    end = clock();

    printf("\nProgramm worked:%f\n", (double)(end-start)/CLOCKS\_PER\_SEC);

    return 0;

}

void\* streamFunc(int \*args[]){

    int len = \*args[0];

    int\* target\_mas = args[1];

    int streams = \*args[2];

    if (streams > 1){

        int len\_1 = 0;

        int len\_2 = 0;

        int returned\_mas\_1[len/2];

        int returned\_mas\_2[len/2];

        int inheritable\_streams\_one = streams/2;

        int inheritable\_streams\_two = streams-inheritable\_streams\_one;

        int\* args\_one[] = {&len\_1, returned\_mas\_1, &inheritable\_streams\_one};

        int\* args\_two[] = {&len\_2, returned\_mas\_2, &inheritable\_streams\_two};

        pthread\_t stream\_one;

        pthread\_t stream\_two;

        for (int index = 1; index < len; index += 2){

            returned\_mas\_1[len\_1] = target\_mas[index-1];

            returned\_mas\_2[len\_2] = target\_mas[index];

            ++len\_1;

            ++len\_2;

        }

        pthread\_create(&stream\_one, NULL, streamFunc, args\_one);

        pthread\_create(&stream\_two, NULL, streamFunc, args\_two);

        pthread\_join(stream\_two, NULL);

        pthread\_join(stream\_one, NULL);

        mergeSort(len\_1, len\_2, returned\_mas\_1, returned\_mas\_2, target\_mas);

    } else {

        recursionSort(len, target\_mas);

    }

}

int recursionSort(int len, int\* mas){

    printf("started\n");

    if (len == 2){

        swapFunc(&mas[0], &mas[1]);

        printf("minimas:%d  %d\n", mas[0], mas[1]);

        return 0;

    } else {

        int len\_1 = len/2;

        int len\_2 = len/2;

        int index\_1 = 0;

        int index\_2 = 0;

        int returned\_mas\_1[len\_1];

        int returned\_mas\_2[len\_2];

        for (int index = 1; index < len; index += 2){

            returned\_mas\_1[index\_1] = mas[index-1];

            returned\_mas\_2[index\_2] = mas[index];

            ++index\_1;

            ++index\_2;

        }

        recursionSort(len\_1, returned\_mas\_1);

        recursionSort(len\_2, returned\_mas\_2);

        printf("ended\n");

        mergeSort(len\_1, len\_2, returned\_mas\_1, returned\_mas\_2, mas);

        printf("sorted\n");

        for (int index = 0; index < len; index++){

            printf("i(%d):%d  ", index, mas[index]);

        }

        printf("\n");

        return 0;

    }

}

void mergeSort(int len\_1, int len\_2, int\* mas\_1, int\* mas\_2, int\* final\_mas){

    int index\_1 = 0;

    int index\_2 = 0;

    int len = len\_1 + len\_2;

    for (int index = 0; index < len; index++){

        printf("ind1:%d  ind2:%d  ind:%d\n", index\_1, index\_2, index);

        if (index\_1 >= len\_1){

            final\_mas[index] = mas\_2[index\_2];

            ++index\_2;

        } else if (index\_2 >= len\_2){

            final\_mas[index] = mas\_1[index\_1];

            ++index\_1;

        } else {

            if (mas\_1[index\_1] < mas\_2[index\_2]){

                final\_mas[index] = mas\_1[index\_1];

                ++index\_1;

            } else {

                final\_mas[index] = mas\_2[index\_2];

                ++index\_2;

            }

        }

    }

}

int swapFunc(int \*elem1, int \*elem2){

    int buffer;

    if (\*elem1 > \*elem2){

        buffer = \*elem1;

        \*elem1 = \*elem2;

        \*elem2 = buffer;

    }

}

**-----------------------------------------------------------**

# **Пример работы**

**Test 1**

|  |  |
| --- | --- |
| Input | Output |
| ./main2 2 |  |

# **Вывод**