Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**УПРАВЛЕНИЕ ПОТОКАМИ В ОС**

Студент: Пермяков Никита Александрович

Группа: М8О–208Б–19

Вариант: 3

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020

**Содержание**

1. Постановка задачи
2. Общие сведения о программе
3. Общий метод и алгоритм решения
4. Основные файлы программы
5. Примеры работы
6. Вывод

**Постановка задачи**

Составить программу, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы. При создании необходимо предусмотреть ключи, которые позволяли бы задать максимальное количество потоков, используемое программой. При возможности необходимо использовать максимальное количество возможных потоков. Ограничение потоков может быть задано или ключом запуска вашей программы, или алгоритмом.

Отсортировать массив строк при помощи параллельной сортировки слиянием.

Базисом для определения метрик параллельных вычислений являются следующие характеристики вычислений:

**n** - количество процессоров, используемых для организации параллельных вычислений;

**O(n)** - объем вычислений, выраженный через количество операций, выполняемых n процессорами в ходе решения задачи;

**Т(n)** - общее время вычислений (решения задачи) с использованием n процессоров.

1. Скорость вычислений:

* Индекс параллелизма: *PI(n)=O(n)/T(n).*
* **Ускорение: *S(n)=T(1)/T(n).***

1. Эффективности привлечения дополнительных процессоров:

* **Эффективность: *E(n)=S(n)/n=T(1)/(n****×****T(n)).***
* Утилизация: *U(n)=R(n)×E(n)=O(n)/(n×T(n)).*

1. Сравнение объема вычислений:

* Избыточность: *R(n)=O(n)/O(1)*.
* Сжатие: *C(n)=O(1)/O(n)*.

**Общие сведения о программе**

Программа компилируется из файла main.cpp. Также используется заголовочные файлы: iostream, string, pthread.h, math.h. В программе используются следующие системные вызовы:

1. **CreateThread** – создает новый поток
2. **pthread\_join** – ожидает завершения переданного потока, получает его выходное значение
3. **pthread\_mutex\_init** – инициализация mutex.
4. **pthread\_mutex\_destroy –** уничтожение mutex.
5. **pthread\_mutex\_lock –** блокировка части кода определенным потоком.
6. **pthread\_mutex\_unlock –** разблокировка части кода определенным потоком.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Ввести размер массива строк
2. Аргументом исполняемого файла задать кол-во потоков (которое приведется к степени двойки (кроме 1).
3. Используя системный вызов fork создать дочерний процесс.
4. Исходный массив сортируется параллельной сортировкой слиянием.
5. Отсортированный массив выводится на стандартный поток вывода.

**Тесты**

Программа поддерживает один аргумент при запуске исполняемого файла – количество создаваемых threads.

Представлено среднее время работы алгоритма, использовалась библиотека: **«sys/time.h»**

Точность подсчета среднего значения до 3 знаков после запятой.

Относительная погрешность: 0.05 %

Абсолютная погрешность: 50 ms

Количество повторов одного теста: **10**

**«Псевдоалгоритм»**

Использование mutex для передачи индекса thread

Первый набор тестов

Входные параметры:

* Количество элементов в массиве: **1`000**
* Диапазон допустимых значений: **[ 0, 1000 )**

**test1: запуск с ключом «5» - 3.697 ms**

**test2: запуск с ключом «15» - 3.835 ms**

**test3: запуск с ключом «25» - 5.945 ms**

**test4: запуск с ключом «1» - 3.032 ms**

**test5: запуск с ключом «2» - 3.102 ms**

Второй набор тестов

Входные параметры:

* Количество элементов в массиве: **100`000**
* Диапазон допустимых значений: **[ 0, 100`000 )**

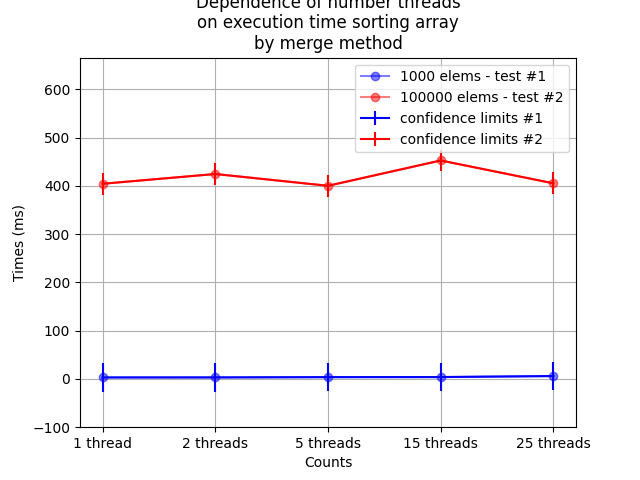
**test1: запуск с ключом «5» - 400.045 ms**

**test2: запуск с ключом «15» - 452.451 ms**

**test3: запуск с ключом «25» - 405.115 ms**

**test4: запуск с ключом «1» - 424.439 ms**

**test5: запуск с ключом «2» - 404.212 ms**



**«Многопоточный алгоритм»**

Использование адресной арифметики для передачи индекса thread

Первый набор тестов

Входные параметры файл gen\_tests.txt:

* Количество элементов в массиве: **1`000`000**
* Диапазон допустимых значений: **[ 0, 1`000 )**

**Test1: запуск с ключом «1» - 0.332334 s**

* Acceleration S(n)=T(1)/T(n): 0.497367
* Efficiency E(n)=S(n)/n=T(1)/(nT(n)): 0.497367

**Test2: запуск с ключом «2» - 0.207382 s**

* Acceleration S(n)=T(1)/T(n): 1.145283
* Efficiency E(n)=S(n)/n=T(1)/(nT(n)): 0.572641

**Test3: запуск с ключом «5» - 0.100345 s**

* Acceleration S(n)=T(1)/T(n): 1.835179
* Efficiency E(n)=S(n)/n=T(1)/(nT(n)): 0.367036

**Test4: запуск с ключом «15» - 0.090462 s**

* Acceleration S(n)=T(1)/T(n): 2.285236
* Efficiency E(n)=S(n)/n=T(1)/(nT(n)): 0.152349

**Test5: запуск с ключом «25» - 0.107350 s**

* Acceleration S(n)=T(1)/T(n): 1.532256
* Efficiency E(n)=S(n)/n=T(1)/(nT(n)): 0.061290

**Test6: запуск без threads - 0.185292 s**

Второй набор тестов

* Количество элементов в массиве: **1`000**
* Диапазон допустимых значений: **[ 0, 1`000`000 )**

**Test1: запуск с ключом «1» - 0.000648 s**

* Acceleration S(n)=T(1)/T(n): 0.316358
* Efficiency E(n)=S(n)/n=T(1)/(nT(n)): 0.316358

**Test2: запуск с ключом «2» - 0.000714 s**

* Acceleration S(n)=T(1)/T(n): 0.140056
* Efficiency E(n)=S(n)/n=T(1)/(nT(n)): 0.070028

**Test3: запуск с ключом «5» - 0.000832 s**

* Acceleration S(n)=T(1)/T(n): 0.146635
* Efficiency E(n)=S(n)/n=T(1)/(nT(n)): 0.029327

**Test4: запуск с ключом «15» - 0.002012 s**

* Acceleration S(n)=T(1)/T(n): 0.058151
* Efficiency E(n)=S(n)/n=T(1)/(nT(n)): 0.003877

**Test5: запуск с ключом «25» - 0.003812 s**

* Acceleration S(n)=T(1)/T(n): 0.027020
* Efficiency E(n)=S(n)/n=T(1)/(nT(n)): 0.001081

**Test6: запуск без threads - 0.000205 s**

**Основные файлы программы**

**main.cpp:**

#include <pthread.h>

#include <sys/time.h>

#include <fstream>

#include <string>

#include <stdlib.h>

#include <chrono>

#include <thread>

long int \* g\_size\_arr = nullptr;

long int \* g\_num\_threads = nullptr;

long int \*arr = nullptr;

long int \*arr\_single\_proc = nullptr;

void merge\_sort(long int\* arr, long int left, long int right);

void merge(long int\* arr, long int left, long int middle, long int right);

void \*thread\_merge\_sort(void\* arg);

void merge\_sections(long int\* arr, long int num\_thread, long int size\_sub\_arr, long int aggregation);

void test\_order(long int\* arr, std::string msg);

int main(int argc, char \*argv[]) {

    g\_num\_threads = new long int;

    \*g\_num\_threads = atoi(argv[1]);

    g\_size\_arr = new long int;

    printf("\n\tCount elements:\t");

    // scanf("%ld", g\_size\_arr);

    \*g\_size\_arr = 1000000;

    arr = new long int[\*g\_size\_arr];

    arr\_single\_proc = new long int[\*g\_size\_arr];

    long int size\_sub\_arr = \*g\_size\_arr / \*g\_num\_threads;

    struct timeval start, end;

    double time\_spent, time\_single\_spent;

    std::ifstream in("gen\_data.txt");

    std::string tmp;

    long int i;

    for (i = 0; i < \*g\_size\_arr; ++i) {

        // arr[i] = std::to\_string(rand() % 100000);

        getline(in, tmp);

        arr[i] = stoi(tmp);

        arr\_single\_proc[i] = arr[i];

    }

    in.close();

    // more proc

    pthread\_t threads[\*g\_num\_threads];

    gettimeofday(&start, NULL);

    for (i = 0; i < \*g\_num\_threads; ++i) {

        int err = pthread\_create(&threads[i], NULL, thread\_merge\_sort, (void \*) i);

        if (err){

            printf("ERROR return code from pthread\_create() is %d\n", err);

            exit(-1);

        }

    }

    for(i = 0; i < \*g\_num\_threads; ++i)

        pthread\_join(threads[i], NULL);

    merge\_sections(arr, \*g\_num\_threads, size\_sub\_arr, 1);

    gettimeofday(&end, NULL);

    time\_spent = ((double) ((double) (end.tv\_usec - start.tv\_usec) / 1000000 + (double) (end.tv\_sec - start.tv\_sec)));

    printf("\n\tTime for %ld proccessing: %f s\n", \*g\_num\_threads, time\_spent);

    test\_order(arr, "multiproccessing");

    // my gun barrel overheated 5s

    std::this\_thread::sleep\_for(std::chrono::milliseconds(3000));

    // single proc

    gettimeofday(&start, NULL);

    merge\_sort(arr\_single\_proc, 0, \*g\_size\_arr);

    gettimeofday(&end, NULL);

    time\_single\_spent = ((double) ((double) (end.tv\_usec - start.tv\_usec) / 1000000 + (double) (end.tv\_sec - start.tv\_sec)));

    printf("\tTime for single proc: %f s\n", time\_single\_spent);

    test\_order(arr\_single\_proc, "singleproccessing");

    // metrics

    printf("\n\n\n\tResults\n");

    printf("\tAcceleration S(n)=T(1)/T(n):         %f\n", time\_single\_spent / time\_spent);

    printf("\tEfficiency E(n)=S(n)/n=T(1)/(nT(n)): %f\n\n", time\_single\_spent / (time\_spent \* (\*g\_num\_threads)));

    printf("\n");

    delete g\_size\_arr;

    delete g\_num\_threads;

    delete[] arr;

    delete[] arr\_single\_proc;

    return 0;

}

void merge(long int\* arr, long int left, long int middle, long int right) {

    long int k = 0;

    long int left\_length = middle - left + 1;

    long int right\_length = right - middle;

    long int left\_array[left\_length];

    long int right\_array[right\_length];

    long int i;

    long int j;

    for (i = 0; i < left\_length; ++i)

        left\_array[i] = arr[left + i];

    for (j = 0; j < right\_length; ++j)

        right\_array[j] = arr[middle + 1 + j];

    i = 0;

    j = 0;

    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;

    }

    while (i < left\_length) {

        arr[left + k] = left\_array[i];

        ++k;

        ++i;

    }

    while (j < right\_length) {

        arr[left + k] = right\_array[j];

        ++k;

        ++j;

    }

}

void merge\_sections(long int\* arr, long int num\_thread, long int size\_sub\_arr, long int aggregation) {

    long int left;

    long int right;

    long int middle;

    for(long int i = 0; i < num\_thread; i = i + 2) {

        left = i \* (size\_sub\_arr \* aggregation);

        right = left + ((i + 2) \* size\_sub\_arr \* aggregation) - 1;

        middle = left + (size\_sub\_arr \* aggregation) - 1;

        if (right >= \*g\_size\_arr) {

            right = \*g\_size\_arr - 1;

        }

        merge(arr, left, middle, right);

    }

    if (num\_thread / 2 >= 1) {

        merge\_sections(arr, num\_thread / 2, size\_sub\_arr, aggregation \* 2);

    }

}

void merge\_sort(long int\* arr, long int left, long int right) {

    long int middle;

    if (left < right) {

        middle = left + (right - left) / 2;

        merge\_sort(arr, left, middle);

        merge\_sort(arr, middle + 1, right);

        merge(arr, left, middle, right);

    }

}

void \*thread\_merge\_sort(void\* arg) {

    long int thread\_id = (long int)arg;

    long int left = thread\_id \* (\*g\_size\_arr / \*g\_num\_threads);

    long int right = (thread\_id + 1) \* (\*g\_size\_arr / \*g\_num\_threads) - 1;

    if (thread\_id == \*g\_num\_threads - 1)

        right += (\*g\_size\_arr % \*g\_num\_threads);

    long int middle = left + (right - left) / 2;

    if (left < right) {

        merge\_sort(arr, left, right);

        merge\_sort(arr, left + 1, right);

        merge(arr, left, middle, right);

    }

    pthread\_exit(nullptr);

}

void test\_order(long int\* arr, std::string msg) {

    long int i;

    bool noerror = true;

    printf("\n\tTests: \n");

    for (i = 1; i < \*g\_size\_arr; ++i) {

        if (arr[i] < arr[i-1]) {

            printf("[ %s ] Error index %ld: prev %ld > current %ld\n", msg.c\_str(), i, arr[i-1], arr[i]);

            noerror = false;

        }

    }

    if (noerror)

        printf("\tArray is in sorted order\n");

    // for (i = 1; i < \*g\_size\_arr; ++i) {

    //     printf("%ld\n", arr[i]);

    // }

}

**Скрипт для получения графика:**

**graph.py**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import matplotlib

df = pd.read\_excel('result\_tests.xlsx', 'Лист1', index\_col=None, na\_values=['NA'])

x = df['time'][1:]

y1 = df['count elements'][1:]

y2 = df['Unnamed: 2'][1:]

y1\_max = max(y1)

y2\_max = max(y2)

y\_max = max(y1\_max, y2\_max)

y1\_absolut\_error = y1\_max \* 0.05

y2\_absolut\_error = y2\_max \* 0.05

# exit()

fig = plt.figure()

ax = fig.add\_subplot(111)

>>>>>>>> для первого набора

ax.plot(x, y1, c='b', marker="o", alpha=0.5, label="1000 elems - test #1")

ax.plot(x, y2, c='r', marker="o", alpha=0.5, label="100000 elems - test #2")

ax.plot(x, y2, c='r', marker="o", alpha=0.5, label="1000000 elems - test #1")

ax.plot(x, y1, c='b', marker="o", alpha=0.5, label="1000 elems - test #2")

<<<<<<<< для второго набора

y1\_err = np.linspace(y1\_absolut\_error, y1\_absolut\_error, len(x))

y2\_err = np.linspace(y2\_absolut\_error, y2\_absolut\_error, len(x))

>>>>>>>> для первого набора

plt.errorbar(x, y1, yerr=y1\_err, color='b', label='confidence lim #1')

plt.errorbar(x, y2, yerr=y2\_err, c='r', label='confidence lim #2')

plt.errorbar(x, y2, yerr=y2\_err, c='r', label='confidence lim #1')

plt.errorbar(x, y1, yerr=y1\_err, color='b', label='confidence lim #2')

<<<<<<<< для второго набора

ax.grid()

ax.set\_title('Dependence of number threads\non execution time sorting array\nby merge method')

plt.ylim(-100, y\_max \* 1.25 + 100)

plt.xlabel("Counts")

>>>>>>>> для первого набора

plt.ylabel("Times (ms)")

plt.ylabel("Times (s)")

<<<<<<<< для второго набора

plt.legend()

plt.savefig('test1.png')

plt.show()

**Скрипт для получения тестов:**

**gen\_tests.py**

import random

with open('gen\_data.txt', 'w') as f:

for count in range(2000000):

num = random.randint(1, 1000)

f.write(str(num))

f.write("\n")

**Вывод**

В результате работы были приобретены навыки работы с многопоточностью, в первом варианте работы использовал mutex для синхронизации потоков. Во втором варианте – арифметика указателей и подсчет границ обработки входного массива для разных процессов. Получены результаты измерений времени работы программы при повторяющихся тестовых данных. Причем использование стандартной библиотеки time.h не привело к результату, так как на выход метод clock выдавал всегда 0. Причиной этому является малый техпроцесс у моего процессора Intel Core i7 1067 10th – составляет 13hm.