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.

**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.

Количество повторов одного теста: **10**

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**

**Основные файлы программы**

**main.cpp:**

#include <iostream>

#include <pthread.h>

#include <string>

#include <cmath>

#include <sys/time.h>

using namespace std;

u\_long g\_size\_arr;

u\_long g\_num\_threads;

pthread\_mutex\_t g\_mutex;

struct params {

string\* array;

u\_long left;

u\_long right;

string\* mod;

};

u\_long char\_to\_int(char c) {

if (c >= '0' && c <= '9') {

c -= '0';

} else if (c >= 'p' && c <= 'z') {

c -= 'W';

} else if (c >= 'A' && c <= 'Z') {

c = tolower(c);

c -= 'W';

}

return c;

}

bool a\_lower\_or\_eq\_b(string a, string b) {

u\_long vec\_a = 0;

u\_long vec\_b = 0;

u\_long k = 1;

u\_long i;

for(i = a.size(); i > 0; --i) {

vec\_a += char\_to\_int(a[i]) \* k;

k \*= 10;

}

k = 1;

for(i = b.size(); i > 0; --i) {

vec\_b += char\_to\_int(b[i]) \* k;

k \*= 10;

}

if (vec\_a <= vec\_b) {

return true;

}

return false;

}

void merge(string \*array, u\_long left, u\_long middle, u\_long right, string \*modif) {

u\_long l = left;

u\_long r = middle;

for (u\_long i = left; i < right; ++i)

if (l < middle && (r >= right || a\_lower\_or\_eq\_b(array[l], array[r])))

modif[i] = array[l++];

else

modif[i] = array[r++];

for (u\_long i = left; i < right; ++i)

array[i] = modif[i];

}

void\* split(void\* param) {

struct params\* temp\_args = new params;

temp\_args = (params\*)param;

if (temp\_args->right - temp\_args->left < 2)

return NULL;

u\_long tmp\_right = temp\_args->right;

u\_long tmp\_left = temp\_args->left;

temp\_args->right = (tmp\_left + tmp\_right) / 2;

split((void\*)temp\_args);

temp\_args->right = tmp\_right;

temp\_args->left = (tmp\_left + tmp\_right) / 2;

split((void\*)temp\_args);

temp\_args->left = tmp\_left;

merge(temp\_args->array, temp\_args->left, (temp\_args->left + temp\_args->right) / 2, temp\_args->right, temp\_args->mod);

return 0;

}

void merge\_sort(string \*array) {

struct params\* p = new params;

string tmp[g\_size\_arr];

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

u\_long new\_left;

u\_long new\_right;

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

new\_left = i \* g\_size\_arr / g\_num\_threads;

new\_right = (i + 1) \* g\_size\_arr / g\_num\_threads;

p->mod = tmp;

p->array = array;

p->left = new\_left;

p->right = new\_right;

pthread\_mutex\_init(&g\_mutex, NULL);

pthread\_create(&threads[i], NULL, split, (void\*)p);

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

pthread\_mutex\_destroy(&g\_mutex);

}

u\_long j;

u\_long left;

u\_long right;

for (u\_long i = g\_num\_threads / 2; i > 0; i = i >> 1) { //divide by 2

for (j = 0; j < i; ++j) {

left = j \* g\_size\_arr / i;

right = (j + 1) \* g\_size\_arr / i;

merge(array, left, (left + right) / 2, right, tmp);

}

}

}

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

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

cout << "\n\tcount elements:\t";

cin >> g\_size\_arr;

// g\_size\_arr = 100000;

string array[g\_size\_arr];

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

cin >> array[i];

// array[i] = to\_string(rand() % 100000);

}

// <<<< time benchmarking

struct timeval tv;

gettimeofday(&tv, NULL);

double time\_begin = ((double)tv.tv\_sec) \* 1000 + ((double)tv.tv\_usec) / 1000;

// >>>> start

u\_long power = 0;

while (g\_num\_threads > 0) {

g\_num\_threads = g\_num\_threads >> 1; // div 2

++power;

}

--power;

if (!power)

power = 1;

g\_num\_threads = (u\_long)pow(2.0, (double)(power));

merge\_sort(array);

cout << "\n Sorted array:\t";

for (u\_long i = 0; i < g\_size\_arr; ++i)

cout << array[i] << " ";

cout << "\n";

// <<<< end

gettimeofday(&tv, NULL);

double time\_end = ((double)tv.tv\_sec) \* 1000 + ((double)tv.tv\_usec) / 1000 ;

double total\_time\_ms = time\_end - time\_begin;

cout << "\n\tTOTAL TIME:\t" << total\_time\_ms << " ms\n";

return 0;

}

**Вывод**

В результате работы были приобретены навыки работы с многопоточностью, использовал mutex для синхронизации потоков. 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 давал всегда 0. Причиной этому является малый техпроцесс у процессоров Intel Core i7 1067 10th – 13hm