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

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