Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**УПРАВЛЕНИЕ ПОТОКАМИ**

Студент: Кудрявов Леонид Вадимович

Группа: М8О–208Б–22

Вариант: 14

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Управление потоками в ОС
* Обеспечение синхронизации между потоками

## Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы 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.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

Вариант 14:

Есть набор 128 битных чисел, записанных в шестнадцатеричном представлении, хранящихся в файле. Необходимо посчитать их среднее арифметическое. Округлить результат до целых. Количество используемой оперативной памяти должно задаваться "ключом".

**Общие сведения о программе**

Программа состоит из файла main.cpp, в котором описан алгоритм выполнения задачи, и файла bigint.cpp, в котором реализованы методы класса для работы с длинными числами. Также используется заголовочный файл: bigint.h. Проект собирается при помощи Cmake.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить как работают потоки в ОС.
2. Ознакомиться с библиотекой pthread.h и mutex для работы с потоками.
3. Считать большие числа из файла в массив строк с учетом ограничений памяти, задаваемое “ключом”.
4. Реализовать функцию для перевода чисел из шестнадцатеричной системы исчисления в десятичную.
5. Реализовать функцию для подсчета суммы чисел в массиве с помощью потоков.
6. Повторять алгоритм в цикле пока не прочитаны все числа из файла.
7. Протестировать работу программы на разном количестве потоков и составить таблицу зависимости ускорения и эффективности алгоритма от входных данных и количества потоков.

**Основные файлы программы**

Main.cpp

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include <cmath>

#include <pthread.h>

#include <mutex>

#include <chrono>

#include "bigint.h"

std::mutex mutex;

typedef struct someArgs\_tag {

int id; // номер потока

int start; // начало диапазона который суммирует поток

int finish; // конец диапазона

} someArgs\_t;

big\_integer from\_16\_to\_10(std::string &arr){

big\_integer res = 0;

for(int i = 0; i < arr.size(); ++i){

if(arr[i] >= '0' and arr[i] <= '9'){

big\_integer st = 16;

st = st.pow(arr.size()-i-1);

st \*= ((int)(arr[i]-'0'));

res += st;

}

else{

int d;

switch (arr[i])

{

case 'A': d = 10; break;

case 'B': d = 11; break;

case 'C': d = 12; break;

case 'D': d = 13; break;

case 'E': d = 14; break;

case 'F': d = 15; break;

}

big\_integer st = 16;

st = st.pow(arr.size()-i-1);

st \*= d;

res += st;

}

}

return res;

}

big\_integer sm = 0; // итоговая сумма чисел из файла

std::vector<std::string> array\_str; // массив для хранения чисел в строчном представлении

void\* sum\_array(void\* args){

someArgs\_t \*arg = (someArgs\_t\*) args;

for(int i = arg->start; i < arg->finish; ++i){

if(i < array\_str.size()){

mutex.lock();

sm += from\_16\_to\_10(array\_str[i]);

mutex.unlock();

}

}

pthread\_exit(0);

}

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

if(argc != 3){

std::cerr << "Key error\n";

return 1;

}

std::string line;

std::string path = "../../test/";

std::string name;

std::cout << "Input test file name: ";

std::cin >> name;

std::cout << std::endl;

auto start\_time = std::chrono::steady\_clock::now();

path += name;

std::ifstream file(path);

const int threadCount = atoi(argv[1]); // количество потоков

int memory = atoi(argv[2]); // в байтах

int global\_arr\_size = 0;

if(memory < 16){

std::cerr << "Less memory\n";

return 1;

}

while(std::getline(file,line)){

array\_str.push\_back(line);

int loc\_memory = 16;

if (file.is\_open()){

while (loc\_memory < memory and std::getline(file, line)){

array\_str.push\_back(line);

loc\_memory += 16; }

}

pthread\_t t[threadCount];

someArgs\_t args[threadCount];

int arr\_size = array\_str.size();

global\_arr\_size += arr\_size;

int step = ((arr\_size) /threadCount) + 1;

if(arr\_size == 0){

std::cerr << "Empty array\n";

break;

}

for(int i{0};i<threadCount;++i){

args[i].id = i;

if(i == 0)

args[i].start = 0;

else

args[i].start = i\*step;

args[i].finish = (i+1)\*step ;

}

for(int i{0};i<threadCount;++i){

pthread\_create(&t[i], NULL, sum\_array, (void\*)&args[i]);

}

for(int i{0};i<threadCount;++i){

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

}

array\_str.clear();

}

file.close();

if(global\_arr\_size == 0){

std::cerr << "Empty file\n";

return 1;

}

big\_integer averg = sm/global\_arr\_size;

std::cout<< averg <<std::endl;

auto end\_time = std::chrono::steady\_clock::now();

auto elapsed\_ns = std::chrono::duration\_cast<std::chrono::nanoseconds>(end\_time - start\_time);

std::cout << elapsed\_ns.count() << " ns\n";

}

**Пример работы**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Вывод**

Мной изучены принципы работы и управления потоками. Во время работы я столкнулся с проблемой: каждый поток считает сумму всего массива и итоговый ответ получался не правильным. Я решил эту проблемы следующим образом: создал структуру, в которой хранил id потока и границы диапазона массива, который данный поток должен суммировать. Также при нескольких запусках с одними и теми же параметрами программа выдавала разные ответы. Это связано с тем, что разные потоки изменяют одну глобальную переменную одновременно. Я исправил данную ошибку с помощью функций mutex.lock() и mutex.unlock() из библиотеки mutex.

Я протестировал работу программы на разных тестах (в одном файле 1000 числе, в другом 11.000 чисел) с разными значениями «ключей» и составил следующие таблицы отражающие зависимости ускорения и эффективности алгоритма от входных данных и количества потоков.

Из данных графиков видно, что значительное ускорение программы происходит до 6-7 потоков, при дальнейшем увеличении числа потоков видимого ускорения работы мы не наблюдаем, при этом большее количество потоков, требует большего количества ресурсов, а следовательно использование такого количества потоков не рационально.