Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №X по курсу**

**«Операционные системы»**

Студент: Савинова Е. И.

Группа: М8О-207Б-21

Вариант: 13

Преподаватель: Черемисинов Максим

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

<https://github.com/savinova-kati/operating-systems>

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

Управление потоками в ОС

Обеспечение синхронизации между потоками

**Задание**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При

обработки использовать стандартные средства создания потоков операционной системы

(Windows/Unix). Ограничение потоков 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 входящих

данных и количества потоков. Получившиеся результаты необходимо объяснить.

Вариант 13) 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 компилируется из файла main.c при помощи cmake. В программе реализована многопоточность. Функции для работы с потоками. которые я использовал:

* pthread\_create() — создание потока с передачей ему аргументов. В случае успеха возвращает 0.
* **pthread\_join()** — ожидает завершения потока обозначенного THREAD\_ID. Если этот поток к тому времени был уже завершен, то функция немедленно возвращает значение.
* mtx.lock(); — блокировка мьютекса
* mtx.unlock(); — открытие доступа к мьютексу

**Общий метод и алгоритм решения**

Требуется найти среднее значение 128 битных 16-ричных чисел. Суть будет в том, что файл с числами разобьётся на количество частей, равное количеству потоков. В каждом потоке будет искаться среднее значение определенной части, а после, складываться к общему среднему, который является глобальной переменной. Количество потоков логично будет ограничить количеством элементов в массиве.

Также, для определения, ускоряет ли работу программы многопоточность, используем chrono::steady\_clock::time\_point

**Исходный код**

#include<pthread.h>

#include<iostream>

#include <ctime>

#include<vector>

#include<fstream>

#include<chrono>

#include<mutex>

#include <string>

#include <sys/time.h>

#include <sys/resource.h>

#include <inttypes.h>

#include <stdlib.h>

//#include <boost/multiprecision/cpp\_int.hpp>

//using namespace boost::multiprecision;

typedef unsigned \_\_int128 int128\_t;

//#define int128\_t int

using namespace std;

int128\_t summ = 0;

int128\_t flag= 1;

mutex mtx;

void setmemlimit(string memory)

{

int mem = stoi(memory);

struct rlimit memlimit;

long bytes;

if(memory!= "")

{

bytes = stol(memory)\*(1024\*1024);

cout << "LIMIT " << stol(memory) << endl;

memlimit.rlim\_cur = bytes;

memlimit.rlim\_max = bytes;

setrlimit(RLIMIT\_AS, &memlimit);

}

}

void translation\_10\_to\_16 (int128\_t number)

{

int iter = 0;

//cout << number << endl;

vector<string> rezult;

while(number != 0) {

if((number % 16) == 0) {

rezult.push\_back("0");

} else if(number % 16 == 1) {

rezult.push\_back("1");

} else if(number % 16 == 2) {

rezult.push\_back("2");

} else if(number % 16 == 3) {

rezult.push\_back("3");

} else if(number % 16 == 4) {

rezult.push\_back("4");

} else if(number % 16 == 5) {

rezult.push\_back("5");

} else if(number % 16 == 6) {

rezult.push\_back("6");

} else if(number % 16 == 7) {

rezult.push\_back("7");

} else if(number % 16 == 8) {

rezult.push\_back("8");

} else if(number % 16 == 9) {

rezult.push\_back("9");

} else if(number % 16 == 10) {

rezult.push\_back("A");

} else if(number % 16 == 11) {

rezult.push\_back("B");

} else if(number % 16 == 12) {

rezult.push\_back("C");

} else if(number % 16 == 13) {

rezult.push\_back("D");

} else if(number % 16 == 14) {

rezult.push\_back("E");

} else if(number % 16 == 15) {

rezult.push\_back("F");

}

//cout << "ITER " << iter << " IS " << rezult[iter] << endl;

//cout << "NUMBER " << number % 16 << endl;

number = number / 16;

iter ++;

}

for(int i = rezult.size() - 1; i > -1; --i) {

cout << rezult[i];

}

}

int128\_t translation\_16\_to\_10(string str)

{

int128\_t stroka = 0;

for(int i = 0; i < str.length(); i++) {

char a;

a = str[i];

//cout << "CHAR " << int(a) << " DEGRE " << str.length() - i - 1 << endl;

if(int(a) == 48) {

stroka += 0;

} else if(int(a) == 49) {

stroka += (1 \* pow(16, str.length() - i - 1));

} else if(int(a) == 50) {

stroka += (2 \* pow(16, str.length() - i - 1));

} else if(int(a) == 51) {

stroka += (3 \* pow(16, str.length() - i - 1));

} else if(int(a) == 52) {

stroka += (4 \* pow(16, str.length() - i - 1));

} else if(int(a) == 53) {

stroka += (5 \* pow(16, str.length() - i - 1));

} else if(int(a) == 54) {

stroka += (6 \* pow(16, str.length() - i - 1));

} else if(int(a) == 55) {

stroka += (7 \* pow(16, str.length() - i - 1));

} else if(int(a) == 56) {

stroka += (8 \* pow(16, str.length() - i - 1));

} else if(int(a) == 57) {

stroka += (9 \* pow(16, str.length() - i - 1));

} else if(int(a) == 65) {

stroka += (10 \* pow(16, str.length() - i - 1));

} else if(int(a) == 66) {

stroka += (11 \* pow(16, str.length() - i - 1));

} else if(int(a) == 67) {

stroka += (12 \* pow(16, str.length() - i - 1));

} else if(int(a) == 68) {

stroka += (13 \* pow(16, str.length() - i - 1));

} else if(int(a) == 69) {

stroka += (14 \* pow(16, str.length() - i - 1));

} else if(int(a) == 70) {

stroka += (15 \* pow(16, str.length() - i - 1));

}

}

//cout << "STRIBG !!!!!!!! " << stroka << endl;

return stroka;

}

struct to\_thread

{

int128\_t\* mas;

int len\_mas;

int number\_of\_threads;

int iterations;

int count\_of\_threads;

} to\_threads;

void\* task(void \*t)

{

int128\_t summ\_1 = 0;

to\_thread\* args = (to\_thread\*) t;

int iterations = args->iterations;

int number\_of\_threads = args->number\_of\_threads;

//cout << "number\_of\_threads " << number\_of\_threads << endl;

flag = 0;

int128\_t\* mas = args->mas;

int count\_of\_threads = args->count\_of\_threads;

int len\_mas = args->len\_mas;

//cout << "LEN " << len\_mas << endl;

//double len\_mas\_2 = len\_mas;

if (number\_of\_threads != count\_of\_threads - 1) {

for(int j = number\_of\_threads \* iterations; j < number\_of\_threads\*iterations + iterations; ++j) {

int128\_t s = mas[j] / len\_mas;

summ\_1 += s;

}

} else {

for(int j = len\_mas - iterations; j < len\_mas; ++j) {

int128\_t s = mas[j] / len\_mas;

summ\_1 += s;

}

}

mtx.lock();

summ += summ\_1;

mtx.unlock();

return 0;

}

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

{

string memory = argv[1];

setmemlimit(memory);

ifstream file("/Users/ekaterina/Desktop/OS/operating-systems/lab3/filename.txt");

int len\_mas = 0;

vector<int128\_t> mas;

while (!file.eof()) {

string str;

int128\_t str2;

getline(file, str);

//cout << "ELEM STR " << str << endl;

str2 = translation\_16\_to\_10(str);

mas.push\_back(str2);

//cout << "ELEM " << mas[len\_mas] << endl;

len\_mas ++;

}

//cout << "LEN " << len\_mas << endl;

int count\_of\_threads;

cout << "Введите количество потоков, которое вы хотите использовать ";

cin >> count\_of\_threads;

cout << endl;

if (count\_of\_threads > len\_mas) {

count\_of\_threads = len\_mas;

}

pthread\_t threads[count\_of\_threads];

int iterations = len\_mas / count\_of\_threads;

struct to\_thread args;

args.len\_mas = len\_mas;

args.count\_of\_threads = count\_of\_threads;

args.mas = mas.data();

chrono::steady\_clock::time\_point begin = chrono::steady\_clock::now();

for (int i = 0; i < count\_of\_threads; i++) {

args.number\_of\_threads = i;

if (i == count\_of\_threads - 1) {

iterations += len\_mas % count\_of\_threads;

}

args.iterations = iterations;

int create = pthread\_create(&threads[i], NULL, task, (void\*)&args);

while(flag != 0) {

}

flag = 1;

if (create != 0) {

cout << "Ошибка в работе потоков" << endl;

}

}

for (int i = 0; i < count\_of\_threads; ++i) {

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

}

chrono::steady\_clock::time\_point end = chrono::steady\_clock::now();

int128\_t rez\_10;

rez\_10 = summ;

//cout << "RES " << rez\_10 << endl;

vector<char> rez\_16;

cout << "ANSWER ";

translation\_10\_to\_16(rez\_10);

cout << endl;

//cout << rez\_10 << endl;

cout << "TIME " << chrono::duration\_cast<chrono::microseconds>(end-begin).count() << endl;

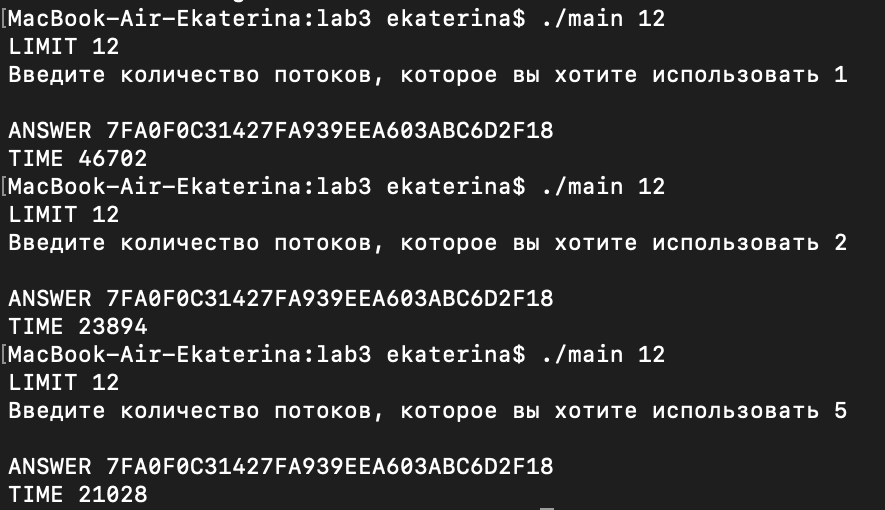
int128\_t t;

return 0;

}

**Демонстрация работы программы**

Продемонстрируем работу многопоточного алгоритма на файле из 10000 чисел

****

Из тестирования видно, что значения при 1, 2, 5 потоках сильно отличаются. Особенно это разница заметна при малом количестве потоков (отличие почти в два раза при 1 и 2 потоках).

**Выводы**

В заключение хотелось бы сказать, что благодаря данной лабораторной работе, я познакомилась с многопоточностью у меня получилось верно реализовать ее для данной задачи. Так же мне удалось выявить зависимость между количеством потоков и временем выполнения. При малом количестве потоков разницы во времени выполнения очень значительна. Дальнейшее увеличение количества потоков приводит лишь к незначительному набору эффективности (много времени тратиться на инициализацию потоков). Я считаю данную работу полезной, так как многопоточность, при программах с большим объемом данных, сильно может сократить затраченное на поиск решения время.