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();
```

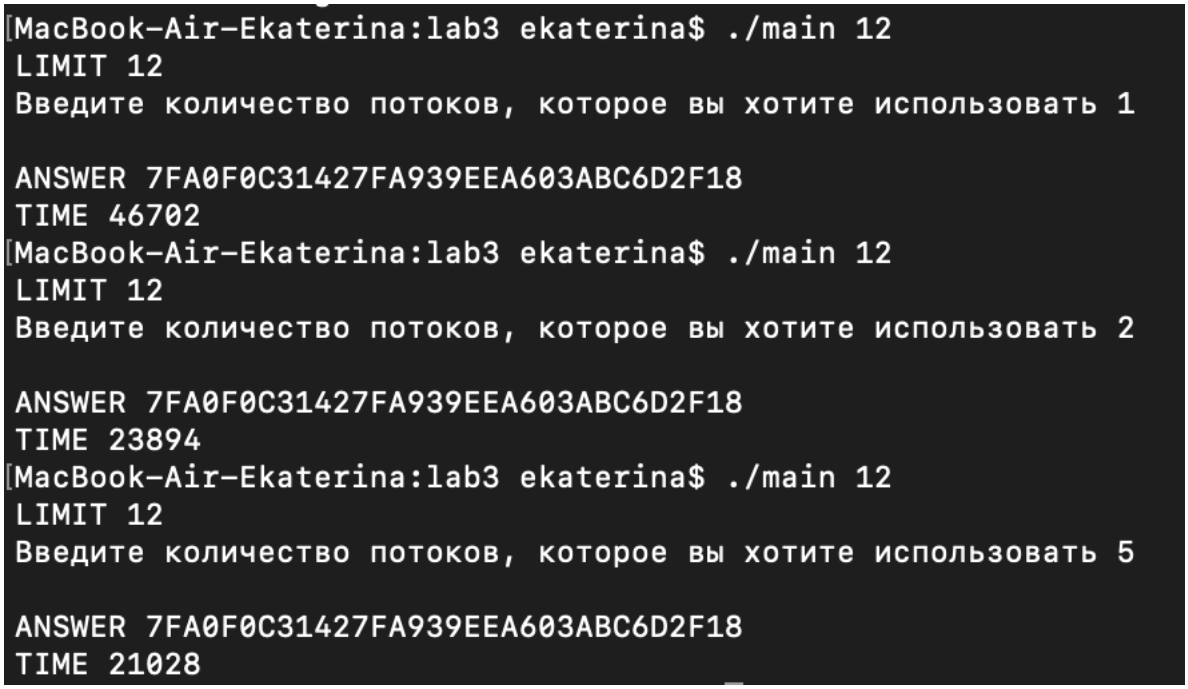
```
                summ += summ_1;


                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.