

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу
«Операционные системы»**

Студент: Мусаелян Ярослав
Группа: М8О-207Б-21
Вариант: 8
Преподаватель: Миронов Евгений Сергеевич
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/YMusaelyan/os>

Постановка задачи

Цель работы

Приобретение практических навыков в управление потоками в ОС и в обеспечение синхронизации между потоками.

Задание

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

8 вариант) Есть K массивов одинаковой длины. Необходимо сложить эти массивы.

Общие сведения о программе

Программа компилируется из файла main.cpp. В программе используются следующие системные вызовы:

- 1 pthread_create() - создает поток.
- 2 pthread_join() - блокирует вызывающий поток, пока указанный поток не завершится.

Общий метод и алгоритм решения

В зависимости от количества потоков, будем разделять наши массивы на отдельные участки, сумму которых будем считать в отдельных потоках и класть в результирующий массив. Например, пусть размеры массивов 7, а потоков 3. Тогда первый поток будет отвечать за сумму всех первых двух элементов, второй так же будет отвечать за сумму следующих двух элементов, а оставшийся третий возьмет оставшиеся три.

Исходный код

main.cpp

```
#include <iostream>
#include <pthread.h>
#include <vector>
#include <chrono>

using namespace std;
using namespace chrono;
```

```

struct Data
{
    vector<vector<int> > &vec;
    vector<int> &res;
    int step;
    int size_arr;
    int k;
    int number_threads;
    int count_threads;
};

void* thread_func(void *argc)
{
    Data* arguments = (Data*) argc;
    vector<vector<int> > &vec = arguments->vec;
    vector<int> &res = arguments->res;
    int step = arguments->step;
    int size_arr = arguments->size_arr;
    int k = arguments->k;
    int number_threads = arguments->number_threads;
    int count_threads = arguments->count_threads;

    if (count_threads != number_threads - 1) {
        for (int i = count_threads * step; i < count_threads * step + step; ++i) {
            for (int j = 0; j < k; ++j) {
                res[i] += vec[j][i];
            }
        }
    } else {
        for (int i = count_threads * step; i < size_arr; ++i) {
            for (int j = 0; j < k; ++j) {
                res[i] += vec[j][i];
            }
        }
    }

    return 0;
}

int main(int argc, const char** argv) {

    if (argc != 2) {
        cerr << "not number of threads\n";
        return 1;
    }

    int number_threads = atoi(argv[1]);
    int size_arr, k;
    cout << "Введите количество массивов\n";
    cin >> k;
    cout << "Введите размер массивов \n";
    cin >> size_arr;

    vector<vector<int> > vec(k, vector<int>(size_arr, 0));
    vector<int> res(size_arr, 0);

    cout << "Введите массивы\n";

```

```

for (int i = 0; i < k; ++i) {
    for (int j = 0; j < size_arr; ++j) {
        cin >> vec[i][j];
    }
}

system_clock::time_point start = system_clock::now();

if (number_threads > size_arr) {
    number_threads = size_arr;
}
int step = size_arr / number_threads;

vector<pthread_t> threads(number_threads);
vector<Data> data(number_threads, {vec, res, step, size_arr, k, number_threads,
0});

for (int i = 0; i < number_threads; i++){
    data[i].count_threads = i;
    if(pthread_create(&threads[i], NULL, thread_func, &data[i]) != 0){
        cerr << "cannot create thread " << i << '\n';
        return 1;
    }
}

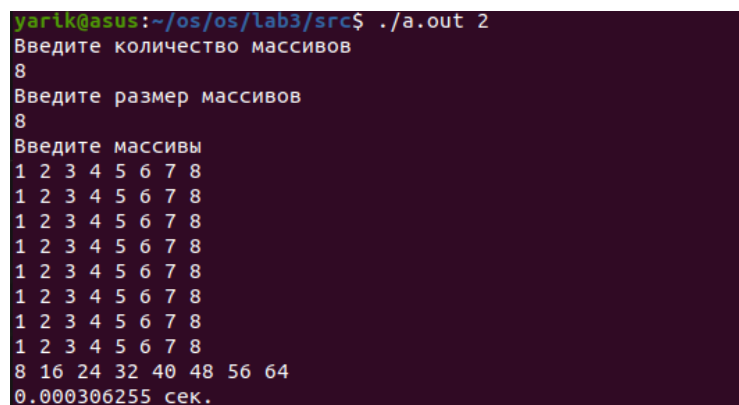
for (int i = 0; i < number_threads; i++){
    if(pthread_join(threads[i], NULL) != 0){
        cerr << "cannot join thread " << i << '\n';
        return 1;
    }
}

system_clock::time_point end = system_clock::now();

for(int i = 0; i < size_arr; ++i) {
    cout << data[number_threads - 1].res[i] << " ";
}
cout << '\n';
duration<double> sec = end - start;
cout << sec.count() << " сек." << endl;
return 0;
}

```

Демонстрация работы программы



```

yarik@gasus:~/os/os/lab3/src$ ./a.out 2
Введите количество массивов
8
Введите размер массивов
8
Введите массивы
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
8 16 24 32 40 48 56 64
0.000306255 сек.

```

Таблица времени для количества потоков и теста:

```
Введите количество массивов
8
Введите размер массивов
8
Введите массивы
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
```

Потоки	Время
1	0.000235042
2	0.000306255
3	0.000360904
4	0.000381201
5	0.000463713
6	0.000588102
7	0.000488338
8	0.000457151
9	0.000548308
10	0.000584275

Выводы

Данная лабораторная работа была очень полезной. Я познакомился с потоками и с управлением потоками в ОС, приобрел практические навыки в управлении потоками в ОС и в обеспечение синхронизации между потоками. В нашем решении мы видим замедление времени с увеличением числа потоков до разделения на 7 потоков. При 7 и 8 потоках мы видим ускорение работы программы, а после 8 снова замедление работы программы. Это связано с тем, что до 7-8 потоков, время на создание потоков больше, чем ускорение, которое они дают при небольшом количестве данных.