Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу
«Операционные системы»

Многопоточные программы

Студент: Бонокин Данил Сергеевич
Группа: М8О–210Б–22
Вариант: 4
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023.

Постановка задачи

Цель работы

Целью является приобретение практических навыков в:

•Написании многопоточных программ.

•Ознакомлении с библиотеками для написания многопоточных программ

Задание

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

Общие сведения о программе

Программа компилируется из файла main.cpp. Также используется заголовочные файлы: vector, iostream, random, thread, chrono . В программе используются следующие системные вызовы:

1. **thread()** –поток, в котором будет выполняться функция
2. **join() –** синхронизирует все потоки

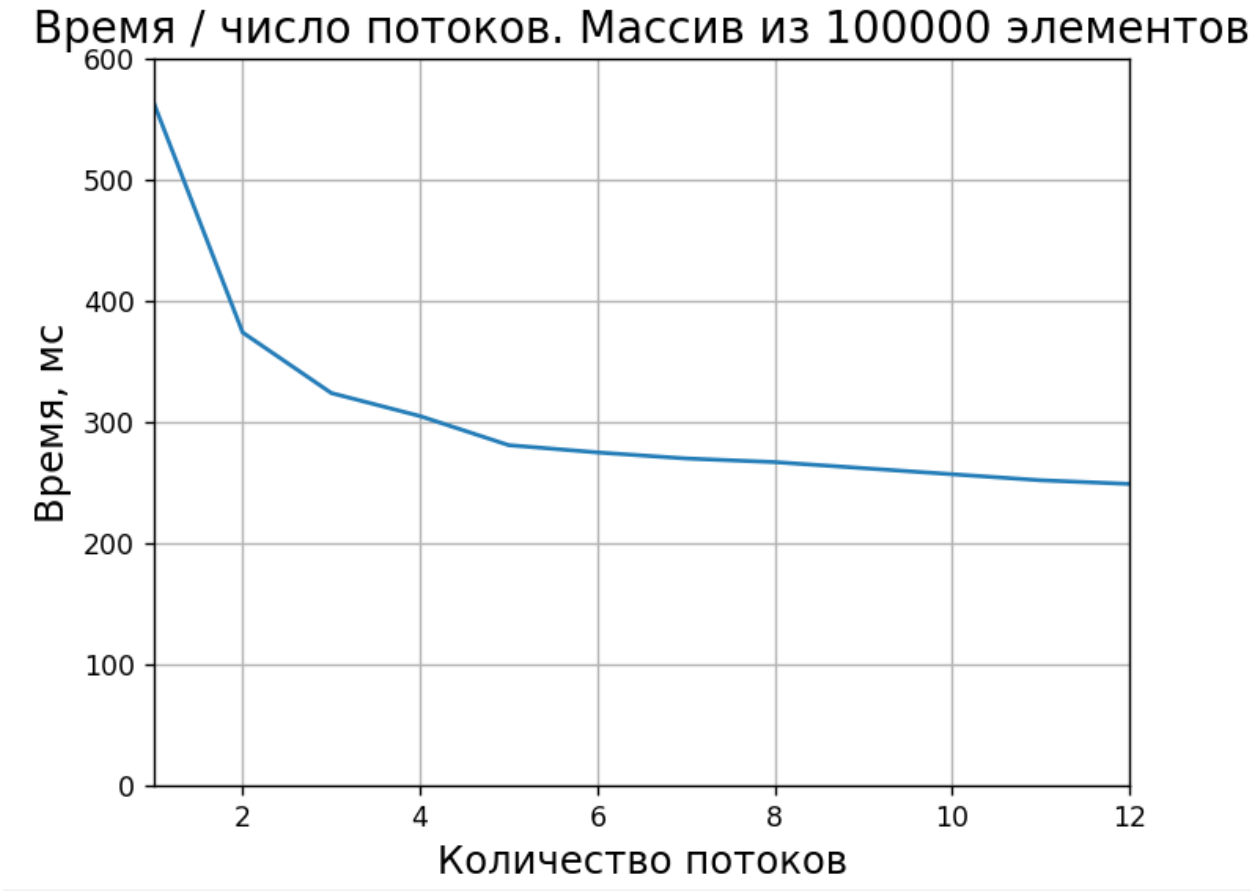
Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

- Изучить принципы работы многопоточных программ.
- Написать асинхронную сортировку
- При компиляции main.cpp указать 1. Размер массива для сортировки 2. Количество потоков

Исследование скорости выполнения программы.

После реализации программы необходимо было проверить, с какой скоростью работает написанный алгоритм при использовании разного максимального количества потоков. Для этого были проведены замеры скорости работы алгоритма при использовании от одного до двенадцати потоков. После этого была составлена таблица, по которой был построен график.



Было замечено, что при увеличении количества потоков время сортировки массива, состоящего из 100000 элементов, каждый раз

уменьшается. Причем выигрыш по времени при использовании двенадцати потоков по сравнению с одним более чем в два раза.

Тесты проводились на процессоре AMD Ryzen 7 5800H with Radeon Graphics, 3201 МГц, ядер: 8, логических процессоров: 16

Основные файлы программы

main.cpp

```
#include <iostream>
#include <vector>
#include <thread>
#include <chrono>
#include <random>


std::vector<int> arrey;
const int RUN = 32;


void insertion_sort(int left, int right){
    for (int i = left + 1; i <= right; i++) {
        int temp = arrey[i];
        int j = i - 1;
        while (j >= left && arrey[j] > temp) {
            arrey[j + 1] = arrey[j];
            j--;
        }
        arrey[j + 1] = temp;
    }
}

void merge(int l, int m, int r){
    int len1 = m - l + 1, len2 = r - m;
```

```
int left[len1], right[len2];
for (int i = 0; i < len1; i++)
    left[i] = arrey[l + i];
for (int i = 0; i < len2; i++)
    right[i] = arrey[m + 1 + i];

int i = 0;
int j = 0;
int k = l;

while (i < len1 && j < len2) {
    if (left[i] <= right[j]) {
        arrey[k] = left[i];
        i++;
    }
    else {
        arrey[k] = right[j];
        j++;
    }
    k++;
}

while (i < len1) {
    arrey[k] = left[i];
    k++;
    i++;
}

while (j < len2) {
    arrey[k] = right[j];
    k++;
    j++;
}
```

```
}

void asinc_sort(int n, int thread_count){
    for (int i = 0; i < n; i += RUN * thread_count){
        std::vector<std::thread> threads;
        for(int j = i; j < i + RUN * thread_count; j += RUN){
            threads.push_back(std::thread(insertion_sort, j, std::min((j + RUN - 1), (n - 1))));
        }
        for (int j = 0; j < threads.size(); j++) {
            threads[j].join();
        }
    }
    for (int size = RUN; size < n; size = 2 * size) {
        std::vector<std::thread> threads;
        for (int left = 0; left < n; left += 2 * size) {

            int mid = left + size - 1;
            int right = std::min((left + 2 * size - 1), (n - 1));

            if (mid < right){
                threads.push_back(std::thread(merge, left, mid, right));
            }
            if (thread_count == threads.size()){
                for(int i = 0; i < thread_count; i++){
                    threads[i].join();
                }
                threads.clear();
            }
        }
        for(int i = 0; i < threads.size(); i++){
            threads[i].join();
        }
    }
```

```
}

int main(int argc, char* argv[]) {
    if (argc < 3) {
        throw std::logic_error("Указано неполное число ключей");
    }
    int n, thread_count;
    thread_count = atoi(argv[1]);
    if (thread_count < 1){
        throw std::logic_error("Количество потоков должно быть больше 1");
    }
    n = atoi(argv[2]);
    if (n < 1){
        throw std::logic_error("Размер массива должен быть больше 1");
    }
    arrey.resize(n);
    for(int i = 0; i < n; i++){
        static std::random_device rd;
        static std::mt19937 gen(rd());
        std::uniform_real_distribution<> dis(0, 100000);
        arrey[i] = dis(gen);
    }
    auto start = std::chrono::steady_clock::now();
    asinc_sort(n, thread_count);
    auto end = std::chrono::steady_clock::now();
    std::cout << std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count();
    for(int i = 0; i < n; i++){
        std::cout << arrey[i] << " ";
    }
}
```

Graph.py

```
import matplotlib.pyplot as plt
import csv
```

```
X = []
Y = []

with open('data.csv', 'r') as datafile:
    plotting = csv.reader(datafile, delimiter=';')

    for ROWS in plotting:
        X.append(float(ROWS[0]))
        Y.append(float(ROWS[1]))

plt.plot(X, Y)
plt.xlim([1, 12])
plt.ylim([0, 600])
plt.ylabel(r'Время, мс', fontsize = 14)
plt.xlabel(r'Количество потоков', fontsize = 14)
plt.title(r'Время / число потоков. Массив из 100000 элементов', fontsize = 16)
plt.grid(True)
plt.show()
```

Пример работы

```
PS C:\Users\Bonik\Desktop> ./a.exe 4 10000
31
PS C:\Users\Bonik\Desktop> ./a.exe 4 100000
294
PS C:\Users\Bonik\Desktop> ./a.exe 12 100000
254
```

```
PS C:\Users\Bonik\Desktop> ./a.exe 12 100000
252
PS C:\Users\Bonik\Desktop> ./a.exe 8 100000
269
```

Вывод

При выполнении данной лабораторной работы я научился писать и отлаживать многопоточные программы. Познакомился с работой библиотеки thread.Было интересно самому столкнуться с написанием многопоточных программ, так как многопоточность имеет множество применений.