**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 3.2**

**по дисциплине «Операционные системы»**

Тема: **«Процессы и потоки»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3311 | Баймухамедов Р. Р. |  |
| Преподаватель | Тимофеев А. В. |  |

Санкт-Петербург

2025

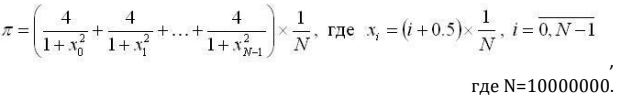
**Цель работы**

Исследовать механизмы создания и управления процессами и потоками в ОС Linux

**Задание**

Постановка задачи и описание решения

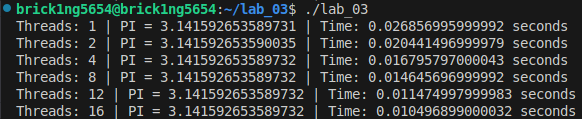
Для выполнения данной лабораторной работы необходимо разработать консольное приложение, которое вычисляет число Пи с точностью N знаков после запятой по следующей формуле:



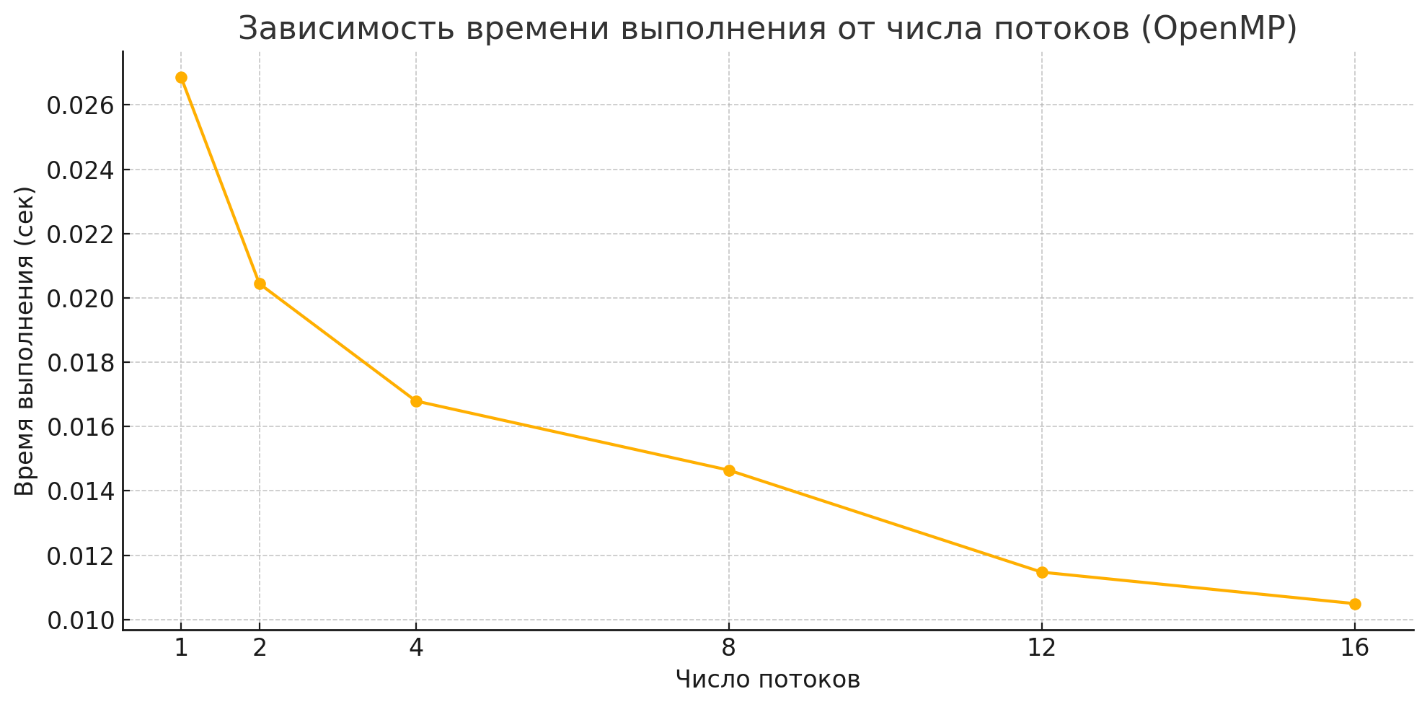
Распределить работу по потокам с помощью OpenMP-директивы for. Использовать динамическое планирование блоками итераций

Провести замеры времени выполнения приложения для разного числа потоков (1, 2, 4, 8, 12 и 16). Сравнить с результатами пункта 3.1.

**Пример**



**График 3**



**Заключение**

Во второй части лабораторной работы было реализовано многопоточное приложение для вычисления числа Пи с использованием технологии OpenMP. Распределение итераций было выполнено с использованием динамического планирования блоками.

По результатам экспериментов с различным числом потоков (1, 2, 4, 8, 12, 16) наблюдается закономерное снижение времени выполнения при увеличении числа потоков. Минимальное время (около 0,010 секунд) достигнуто при 16 потоках. Общая динамика ускорения аналогична реализации на Win32 API, однако производительность с OpenMP оказалась в среднем выше. Например, при 16 потоках WinAPI-реализация показала 0,013 секунды, тогда как OpenMP — около 0,010 секунд.

Это объясняется тем, что OpenMP эффективно управляет распределением работы между потоками, снижая накладные расходы, связанные с ручным управлением состояниями потоков (Suspend/Resume).

Сравнение полученных графиков для реализаций на Win32 API и OpenMP показало, что общая форма графиков практически совпадает: в обоих случаях наблюдается быстрое уменьшение времени при увеличении числа потоков до 8, после чего график выходит на плато. Это подтверждает, что вычислительная нагрузка эффективно распараллеливается до определённого предела, ограниченного архитектурой процессора и накладными расходами на переключение контекста.

**Код программы**

#include <iostream>

#include <omp.h>

using namespace std;

const int N = 10000000;

const int BLOCK\_SIZE = 3311030; // 331103 - numb of stud bilet

int main() {

cout.precision(15);

cout << fixed;

int thread\_counts[] = {1, 2, 4, 8, 12, 16};

for (int threads : thread\_counts) {

double pi = 0.0;

double start\_time = omp\_get\_wtime();

omp\_set\_num\_threads(threads);

#pragma omp parallel for schedule(dynamic, BLOCK\_SIZE) reduction(+:pi)

for (int i = 0; i < N; ++i) {

double x = (i + 0.5) / N;

pi += 4.0 / (1.0 + x \* x);

}

pi = pi/N;

double end\_time = omp\_get\_wtime();

cout << "Threads: " << threads << " | PI = " << pi << " | Time: " << (end\_time - start\_time) << " seconds" << endl;

}

return 0;

}