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:

$$\pi = \left(\frac{4}{1+x_0^2} + \frac{4}{1+x_1^2} + \ldots + \frac{4}{1+x_{N-1}^2} \right) \times \frac{1}{N}, \text{ где } x_i = (i+0.5) \times \frac{1}{N}, \; i = \overline{0, N-1}$$

где N=10000000.

Распределить работу по потокам с помощью OpenMP-директивы for. Использовать динамическое планирование блоками итераций, где размер блока равен 3311030.

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

Пример работоспособности приложения

```
brick1ng5654@brick1ng5654:~/lab_03$ ./lab_03
Threads: 1 | PI = 3.141592653589731 | Time: 0.026856995999992 seconds
Threads: 2 | PI = 3.141592653590035 | Time: 0.020441496999979 seconds
Threads: 4 | PI = 3.141592653589732 | Time: 0.016795797000043 seconds
Threads: 8 | PI = 3.141592653589732 | Time: 0.014645696999992 seconds
Threads: 12 | PI = 3.141592653589732 | Time: 0.011474997999983 seconds
Threads: 16 | PI = 3.141592653589732 | Time: 0.010496899000032 seconds
```

График 3.2

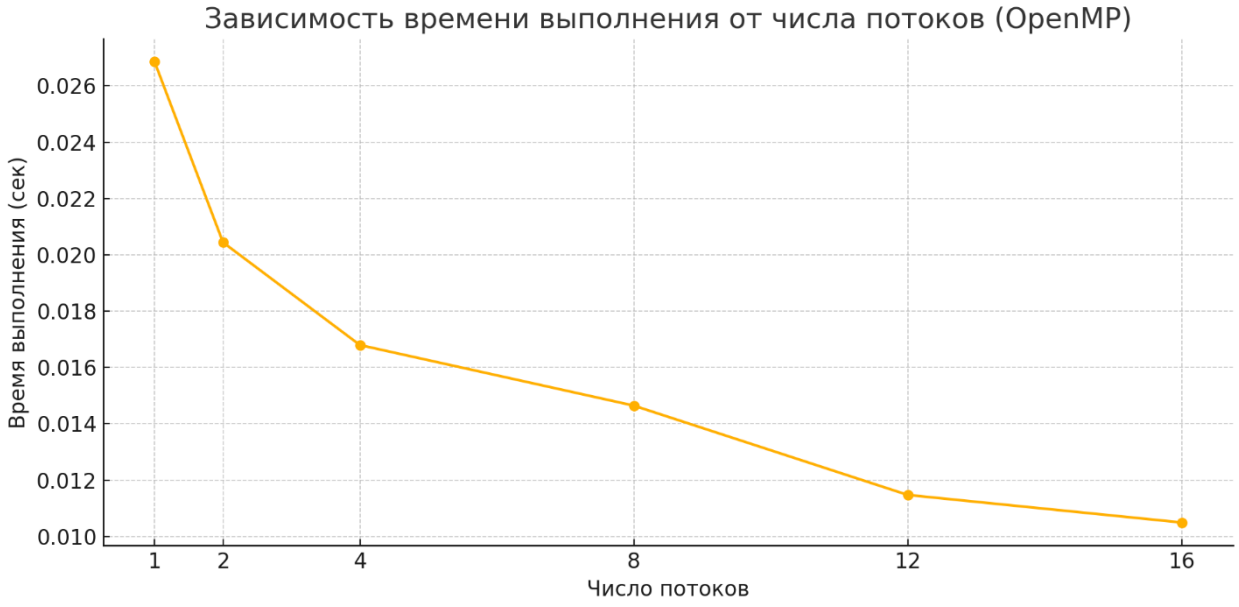
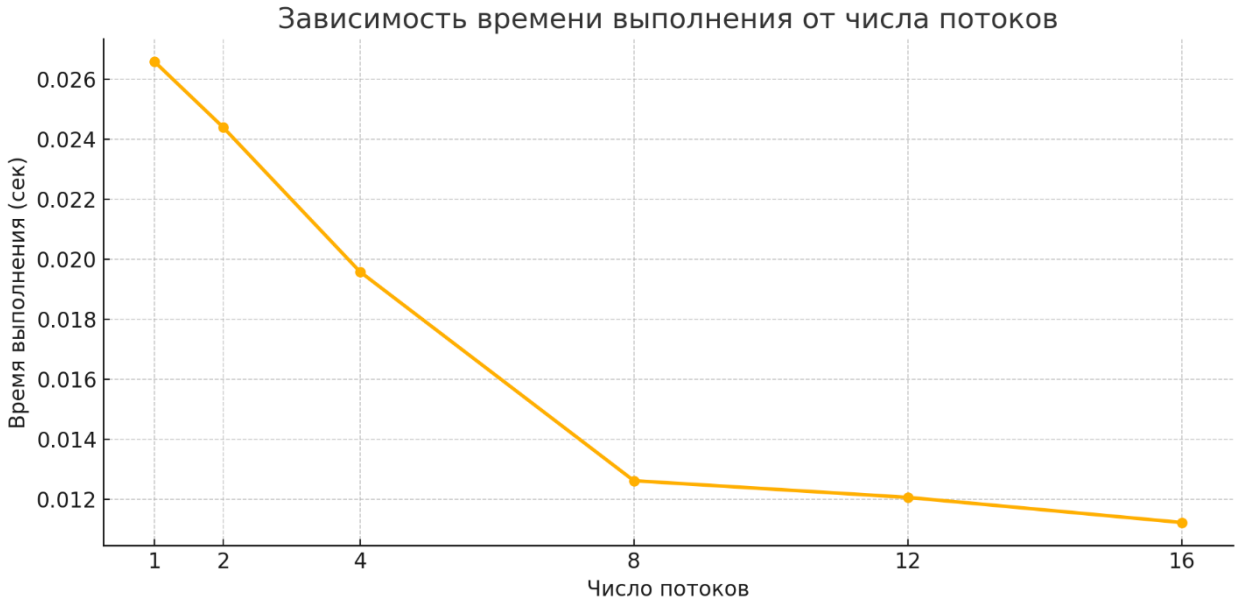


График 3.1



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 размером 3311030 итераций.

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;
}
```