**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

**отчёт   
ЛАБОРАТОРНАЯ РАБОТА № 3**

**по дисциплине «Операционные Системы»**

**Тема: Процессы и потоки**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3311 |  | Сапронов К.Д. |
| Преподаватель |  | Тимофеев А.В. |

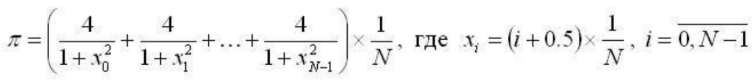
Санкт-Петербург

2025

**Цель работы:** Исследование механизмов создания и управления процессами и потоками в ОС Windows.

**Задание 3.1. Реализация многопоточного приложения с использованием функций Win32 API.**

1. Создайте приложение, которое вычисляет число pi с точностью N знаков после запятой по следующей формуле:

,

где N = 100000000

1. Произведите замеры времени выполнения приложения

для разного числа потоков (1, 2, 4, 8, 12, 16). По результатам

измерений постройте график и определите число потоков, при

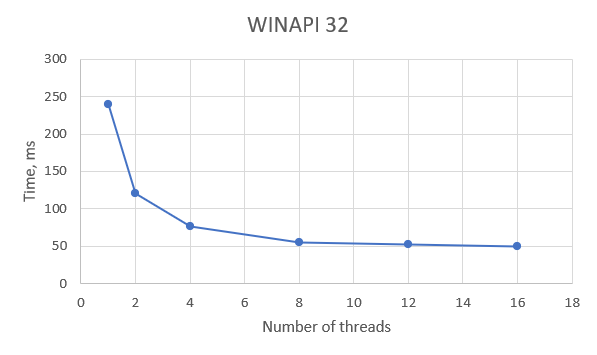
котором достигается наибольшая скорость выполнения.

**Выполнение задания**

Для выполнения была написана программа, которая использует WinAPI для ручного управления потоками. Результаты вычислений суммируются в переменной pi\_result. Также в начале создается массив дескрипторов потоков thread\_pool. Каждому потоку передается структура ThreadData, содержащая его ID, начальный и конечный индекс блока данных. Главный поток создает рабочие потоки в приостановленном состоянии, назначает им начальные блоки и запускает их.

Каждый поток вычисляет свою часть суммы по формуле. После завершения блока поток блокирует мьютекс, обновляет pi\_result, помечает себя как свободный в массиве is\_working, освобождает мьютекс и приостанавливается. Главный поток в цикле проверяет is\_working и назначает свободным потокам новые блоки, увеличивая next\_block. Когда все блоки обработаны, потоки получают команду на завершение, главный поток дожидается их окончания через WaitForMultipleObjects и выводит результат.

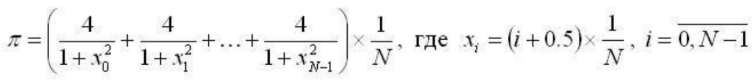
Полученные рещультаты на графике ниже:



Наиболее оптимальным количеством потоков можно считать 8, так так последующее увеличение не дает значительного увеличения производительности.

**Задание 1.2. Реализация многопоточного приложения с использованием технологии OpenMP.**

1. Создайте приложение, которое вычисляет число pi с точностью N знаков после запятой по следующей формуле:

,

где N = 100000000

1. Произведите замеры времени выполнения приложения

для разного числа потоков (1, 2, 4, 8, 12, 16). По результатам

измерений постройте график и определите число потоков, при

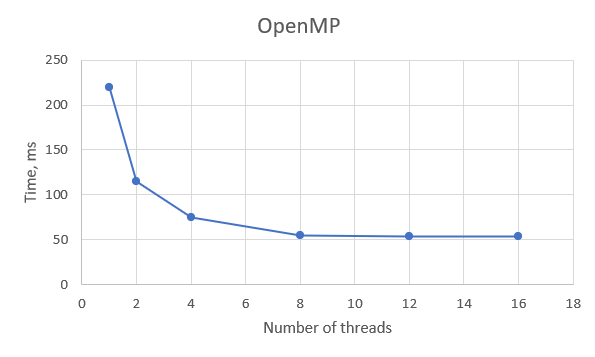
котором достигается наибольшая скорость выполнения, сравните с результатами прошлой работы.

**Выполнение задания**

Программа использует OpenMP для автоматического распараллеливания. Пользователь вводит количество потоков, которое задается через thread\_count. Основные вычисления выполняются в цикле с директивой #pragma omp parallel for, которая автоматически распределяет итерации между потоками. Параметр schedule(dynamic, BLOCK\_SIZE) обеспечивает динамическое распределение блоков, а reduction(+:pi\_result) автоматически суммирует результаты всех потоков.

OpenMP самостоятельно управляет созданием потоков, распределением работы и синхронизацией, что значительно сокращает объем кода по сравнению с WinAPI.

График:



Наиболее выгодным также является вариант с 8 потоками, так как дальнейшее увеличение не дает прироста к скорости.

**Выводы:**

В ходе работы были изучены два подхода к реализации многопоточных вычислений – с использованием WinAPI и OpenMP. Оба метода показали схожую производительность: время выполнения программ практически совпало при одинаковом количестве потоков, а оптимальным числом потоков в обоих случаях оказалось 8. Реализация на OpenMP потребовала значительно меньшего объема кода благодаря автоматическому распределению нагрузки и встроенным механизмам синхронизации. В отличие от ручного управления потоками через WinAPI, где требовалось явно создавать и контролировать потоки, а также обеспечивать безопасный доступ к общим данным через мьютексы, OpenMP позволил достичь аналогичной эффективности с меньшими усилиями.