**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 3.1**

**по дисциплине «Операционные системы»**

Тема: **«Процессы и потоки»**

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

Санкт-Петербург

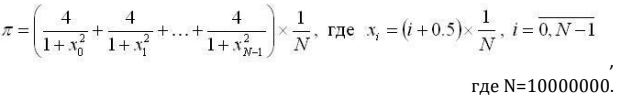
2025

**Цель работы**

Исследовать механизмы создания и управления процессами и потоками в ОС Windows

**Задание**

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



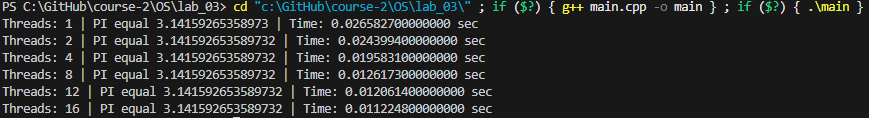
Использовать распределение итераций блоками по потокам. Сначала каждый поток по очереди получает свой блок итераций, затем тот поток, который заканчивает выполнение своего блока, получает следующий свободный блок итераций. Освободившиеся потоки получают новые блоки итераций до тех пор, пока все блоки не будут исчерпаны.

Для реализации механизма распределения блоков итераций необходимо сразу в начале программы создать необходимое количество потоков в приостановленном состоянии.

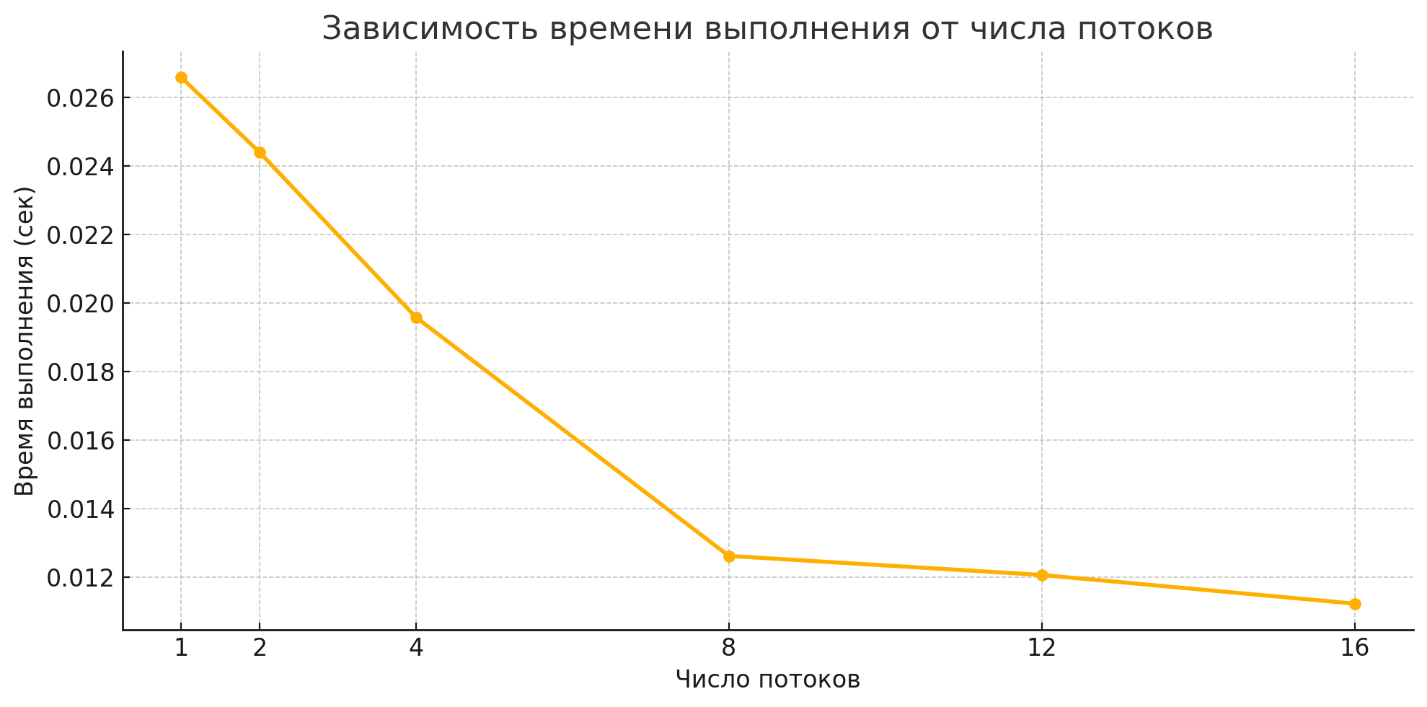
По окончании обработки текущего блока итераций поток не должен завершаться, а должен быть приостановлен. Затем потоку должна быть предоставлен следующий свободный блок итераций, и поток должен быть освобожден.

Провести замеры времени выполнения приложения для разного числа потоков (1, 2, 4, 8, 12 и 16)

**Пример**



**График**



**Заключение**

В данной лабораторной работе разработано многопоточное приложение для вычисления числа Пи с точностью N=10000000 знаков после запятой. Каждый поток брал блок, обрабатывал его, приостанавливался и получал новый свободный блок.

На графике видно, что время вычисления числа Пи уменьшается с 0,026 секунд при работе одного потока до 0,012 секунд при шестнадцати потоках. Однако начиная с 8 потоков прирост производительности замедляется и остается примерно на уровне 0,012 секунд. Это происходит из-за ограничений процессора компьютер и накладных расходов на многопоточность.

**Код программы**

#include <windows.h>

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

const int N = 10000000;

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

const int TOTAL\_BLOCKS = (N + BLOCK\_SIZE - 1) / BLOCK\_SIZE;

struct thread\_data {

    int thread\_id;

};

double pi = 0.0;

int current\_block = 0;

CRITICAL\_SECTION pi\_cs;

CRITICAL\_SECTION block\_cs;

HANDLE\* threads;

thread\_data\* thread\_info;

int num\_threads;

DWORD WINAPI compute\_pi(LPVOID param) {

    thread\_data\* data = static\_cast<thread\_data\*>(param);

    while (true) {

        SuspendThread(GetCurrentThread());

        int block\_id;

        EnterCriticalSection(&block\_cs);

        block\_id = current\_block++;

        LeaveCriticalSection(&block\_cs);

        if (block\_id >= TOTAL\_BLOCKS) break;

        int start = block\_id \* BLOCK\_SIZE;

        int end = min(start + BLOCK\_SIZE, N);

        double local\_sum = 0.0;

        for (int i = start; i < end; ++i) {

            double x = (i + 0.5) / N;

            local\_sum += 4.0 / (1.0 + x \* x);

        }

        EnterCriticalSection(&pi\_cs);

        pi += local\_sum;

        LeaveCriticalSection(&pi\_cs);

    }

    return 0;

}

int main() {

    InitializeCriticalSection(&pi\_cs);

    InitializeCriticalSection(&block\_cs);

    vector<int> thread\_counts = {1, 2, 4, 8, 12, 16};

    for (int count : thread\_counts) {

        pi = 0.0;

        current\_block = 0;

        num\_threads = count;

        threads = new HANDLE[num\_threads];

        thread\_info = new thread\_data[num\_threads];

        for (int i = 0; i < num\_threads; ++i) {

            thread\_info[i].thread\_id = i;

            threads[i] = CreateThread(NULL, 0, compute\_pi, &thread\_info[i], CREATE\_SUSPENDED, NULL);

        }

        LARGE\_INTEGER frequency, start, end;

        QueryPerformanceFrequency(&frequency);

        QueryPerformanceCounter(&start);

        for (int i = 0; i < num\_threads; ++i) {

            ResumeThread(threads[i]);

        }

        bool all\_done = false;

        while (!all\_done) {

            all\_done = true;

            for (int i = 0; i < num\_threads; ++i) {

                DWORD code;

                GetExitCodeThread(threads[i], &code);

                if (code == STILL\_ACTIVE) {

                    ResumeThread(threads[i]);

                    all\_done = false;

                }

            }

        }

        WaitForMultipleObjects(num\_threads, threads, TRUE, INFINITE);

        QueryPerformanceCounter(&end);

        double elapsed\_time = static\_cast<double>(end.QuadPart - start.QuadPart) / frequency.QuadPart;

        pi /= N;

        cout.precision(15);

        cout << "Threads: " << num\_threads << " | PI equal " << pi << " | Time: " << fixed << elapsed\_time << " sec" << endl;

        for (int i = 0; i < num\_threads; ++i)

            CloseHandle(threads[i]);

        delete[] threads;

        delete[] thread\_info;

    }

    DeleteCriticalSection(&pi\_cs);

    DeleteCriticalSection(&block\_cs);

    return 0;

}