**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе №3**

**по дисциплине**

**«Операционные системы»**

Тема: **Процессы и потоки**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3316 |  | Кудрявцев А.А. |
| Преподаватель |  | Тимофеев А.В. |

Санкт-Петербург

2025

**Задание 1.1. Реализация многопоточного приложения с использованием функций Win32 API.**

**Цель работы**: исследовать механизмы создания и управления процессами и потоками в ОС Windows.

**Описание алгоритма:**

Для приближённого вычисления числа π используется метод прямоугольников (интегральное приближение):

****

**1. Глобальные переменные:**

* pi — общее значение приближённого π, накапливаемое из частичных сумм.
* current\_iteration — счётчик текущей позиции в общем числе итераций (обновляется потоками).
* mutex — объект синхронизации (HANDLE), обеспечивающий взаимное исключение доступа к разделяемым данным (pi, current\_iteration).

**2. Константы:**

* N = 10 000 000 — общее количество итераций
* BLOCK\_SIZE = 10 \* 3316 — блок итераций, которые берёт поток за один заход (число получено из номера студенческого билета)

**3. Функция CalculatePi (выполняется каждым потоком):**

* Поток захватывает mutex и получает диапазон итераций (start и end).
* Вычисляет локальную частичную сумму на своём участке (без использования глобальных переменных).
* После завершения вычислений поток снова захватывает mutex, чтобы безопасно прибавить результат к общей переменной pi.
* Цикл продолжается, пока не будут исчерпаны все N итераций.

**4. main():**

* Сначала замеряется время выполнения с помощью QueryPerformanceCounter и QueryPerformanceFrequency.
* Получается количество потоков (здесь — 4), выделяется память под их дескрипторы.
* Создаётся mutex и инициализируется счётчик current\_iteration.
* Запускаются потоки с помощью CreateThread, и каждый выполняет CalculatePi.
* После завершения всех потоков (WaitForMultipleObjects) итоговое значение pi делится на N.
* Результат и время выполнения выводятся на экран.
* Все ресурсы (дескрипторы потоков и mutex) освобождаются.

Функция CalculatePi объявлена как возвращающая тип DWORD по причине того, что функция потока в Win32 API должна соответствовать строго определённой сигнатуре:

DWORD WINAPI ThreadFunction(LPVOID lpParam);

Строка WaitForSingleObject(mutex, INFINITE); выполняет блокировку потока, пока mutex не станет доступным. Это предотвращает гонку данных.

|  |  |
| --- | --- |
| Количество потоков | Время выполнения |
| 1 | 0,013 |
| 2 | 0,010 |
| 4 | 0,005 |
| 8 | 0,004 |
| 12 | 0,004 |
| 16 | 0,004 |

**Пример выполнения программы**



**Текст программы**

#include <windows.h>

#include <stdio.h>

#include <stdlib.h>

#define N 10000000

#define BLOCK\_SIZE (3316130)

double pi = 0.0;

int current\_iteration = 0;

HANDLE mutex;

DWORD WINAPI CalculatePi(LPVOID param) {

    double local\_pi = 0.0;

    int start, end;

    while (1) {

        WaitForSingleObject(mutex, INFINITE);

        start = current\_iteration;

        end = (current\_iteration += BLOCK\_SIZE);

        ReleaseMutex(mutex);

        if (start >= N) break;

        if (end > N) end = N;

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

            double x = (i + 0.5) \* (1.0 / N);

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

        }

    }

    WaitForSingleObject(mutex, INFINITE);

    pi += local\_pi;

    ReleaseMutex(mutex);

    return 0;

}

int main() {

    LARGE\_INTEGER freq, start, end;

    QueryPerformanceFrequency(&freq);

    QueryPerformanceCounter(&start);

    SYSTEM\_INFO sysinfo;

    GetSystemInfo(&sysinfo);

    int num\_threads = 4;

    HANDLE\* threads = (HANDLE\*)malloc(num\_threads \* sizeof(HANDLE));

    mutex = CreateMutex(NULL, FALSE, NULL);

    current\_iteration = 0;

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

        threads[i] = CreateThread(NULL, 0, CalculatePi, NULL, 0, NULL);

        if (!threads[i]) {

            printf("Ошибка создания потока %d\n", i);

            return 1;

        }

    }

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

    pi /= N;

    QueryPerformanceCounter(&end);

    double time = (double)(end.QuadPart - start.QuadPart) / freq.QuadPart;

    printf("Calculated pi = %.15f\n", pi);

    printf("Execution time: %.3f seconds\n", time);

    printf("Threads used: %d\n", num\_threads);

    for (int i = 0; i < num\_threads; i++) CloseHandle(threads[i]);

    free(threads);

    CloseHandle(mutex);

    return 0;

}

**Выводы**

**Анализ результатов:**

Основные оптимизации:

Уменьшение contention на мьютексе:

* + Каждый поток теперь накапливает свою частичную сумму локально (в переменной local\_pi)
  + Глобальный мьютекс используется только для получения нового блока работы
  + Финальное суммирование происходит после завершения всех потоков

Эти изменения должны значительно улучшить производительность, особенно на многоядерных процессорах, за счет уменьшения contention на общих ресурсах.

**Анализ результатов:**

* При увеличении количества потоков от 1 до 4 наблюдается заметное уменьшение времени выполнения, что соответствует ожидаемому эффекту от многопоточности.
* При дальнейшем увеличении числа потоков до 8 — время не улучшается, но остаётся на том же уровне, что свидетельствует о достижении предела эффективности параллелизма для данной задачи или аппаратной конфигурации (например, количества ядер).
* Результаты показывают, что оптимальным числом потоков для данной задачи является 4, при большем количестве наблюдается падение эффективности.

В данной работе реализовано эффективное многопоточное вычисление числа π с использованием низкоуровневых средств Windows API. Программа демонстрирует базовые приёмы работы с потоками, синхронизацией и точными таймерами, что важно для системного программирования. Методика может применяться для решения широкого класса задач с большим количеством итеративных операций.