# Цель и задачи лабораторной работы

**Цель:** изучить механизм работы с классами задач.

**Задачи:**

1. Изучить возможности класса Task;
2. Научиться создавать и запускать задачи;
3. Научиться работать с массивами задач.

# Реализация индивидуального задания

Согласно варианту задания, требуется реализовать с использованием класса Task:

Метод вычисления скалярного произведения двух случайных векторов.

## Листинг программного кода

***namespace*** lab8

{

**using** System;

**using** System.Linq;

**using** System.Threading.Tasks;

***class*** Program

    {

*// Метод генерации случайного вектора*

**static** **double**[] GenerateRandomVector(**int** size, **double** minValue **=** 0, **double** maxValue **=** 10)

        {

            Random random **=** **new**();

**double**[] vector **=** **new** **double**[size];

*for* (**int** i **=** 0; i **<** size; i**++**)

            {

                vector[i] **=** minValue **+** (random.NextDouble() **\*** (maxValue **-** minValue));

            }

*return* vector;

        }

*// Синхронная версия скалярного произведения*

**static** **double** DotProductSync(**double**[] vector1, **double**[] vector2)

        {

*if* (vector1.Length **!=** vector2.Length)

*throw* **new** ArgumentException("Векторы должны иметь одинаковую длину");

**double** result **=** 0;

*for* (**int** i **=** 0; i **<** vector1.Length; i**++**)

            {

                result **+=** vector1[i] **\*** vector2[i];

            }

*return* result;

        }

*// Асинхронная версия с использованием Task*

**static** **async** Task<**double**> DotProductAsync(**double**[] vector1, **double**[] vector2)

        {

*if* (vector1.Length **!=** vector2.Length)

*throw* **new** ArgumentException("Векторы должны иметь одинаковую длину");

*// Запускаем вычисление в отдельной задаче*

*return* **await** Task.Run(() **=>**

            {

**double** result **=** 0;

*for* (**int** i **=** 0; i **<** vector1.Length; i**++**)

                {

                    result **+=** vector1[i] **\*** vector2[i];

                }

*return* result;

            });

        }

*// Параллельная версия с разбиением на части*

**static** **async** Task<**double**> DotProductParallelAsync(**double**[] vector1, **double**[] vector2, **int** partitions **=** 4)

        {

*if* (vector1.Length **!=** vector2.Length)

*throw* **new** ArgumentException("Векторы должны иметь одинаковую длину");

**int** vectorLength **=** vector1.Length;

**int** partitionSize **=** vectorLength **/** partitions;

*// Создаем задачи для каждой части векторов*

***var*** tasks **=** **new** Task<**double**>[partitions];

*for* (**int** i **=** 0; i **<** partitions; i**++**)

            {

**int** start **=** i **\*** partitionSize;

**int** end **=** (i **==** partitions **-** 1) **?** vectorLength **:** start **+** partitionSize;

*// Захватываем переменные для каждой итерации*

**int** localStart **=** start;

**int** localEnd **=** end;

                tasks[i] **=** Task.Run(() **=>**

                {

**double** partialSum **=** 0;

*for* (**int** j **=** localStart; j **<** localEnd; j**++**)

                    {

                        partialSum **+=** vector1[j] **\*** vector2[j];

                    }

*return* partialSum;

                });

            }

*// Ждем завершения всех задач и суммируем результаты*

**double**[] partialResults **=** **await** Task.WhenAll(tasks);

*return* partialResults.Sum();

        }

**public** **static** **async** Task Main(**string**[] args)

        {

**const** **int** vectorSize **=** 1000000; *// Большой размер для демонстрации*

            Console.WriteLine("Генерация случайных векторов...");

**double**[] vector1 **=** GenerateRandomVector(vectorSize);

**double**[] vector2 **=** GenerateRandomVector(vectorSize);

            Console.WriteLine($"Размер векторов: {vectorSize} элементов");

*// Тестирование синхронной версии*

            Console.WriteLine("\n1. Синхронное вычисление:");

***var*** syncResult **=** DotProductSync(vector1, vector2);

            Console.WriteLine($"Результат: {syncResult**:**F6}");

*// Тестирование асинхронной версии*

            Console.WriteLine("\n2. Асинхронное вычисление:");

***var*** asyncResult **=** **await** DotProductAsync(vector1, vector2);

            Console.WriteLine($"Результат: {asyncResult**:**F6}");

*// Тестирование параллельной версии*

            Console.WriteLine("\n3. Параллельное вычисление (4 части):");

***var*** parallelResult **=** **await** DotProductParallelAsync(vector1, vector2, 4);

            Console.WriteLine($"Результат: {parallelResult**:**F6}");

*// Сравнение производительности*

            Console.WriteLine("\n4. Сравнение производительности:");

***var*** watch **=** System.Diagnostics.Stopwatch.StartNew();

            syncResult **=** DotProductSync(vector1, vector2);

            watch.Stop();

            Console.WriteLine($"Синхронная версия: {watch.ElapsedMilliseconds} мс");

            watch.Restart();

            asyncResult **=** **await** DotProductAsync(vector1, vector2);

            watch.Stop();

            Console.WriteLine($"Асинхронная версия: {watch.ElapsedMilliseconds} мс");

            watch.Restart();

            parallelResult **=** **await** DotProductParallelAsync(vector1, vector2, 4);

            watch.Stop();

            Console.WriteLine($"Параллельная версия: {watch.ElapsedMilliseconds} мс");

*// Демонстрация с меньшими векторами для проверки правильности*

            Console.WriteLine("\n5. Проверка правильности (малые векторы):");

**double**[] small1 **=** [1, 2, 3];

**double**[] small2 **=** [4, 5, 6];

***var*** testResult **=** **await** DotProductAsync(small1, small2);

            Console.WriteLine($"Вектор 1: [{**string**.Join(", ", small1)}]");

            Console.WriteLine($"Вектор 2: [{**string**.Join(", ", small2)}]");

            Console.WriteLine($"Скалярное произведение: {testResult} (ожидалось: 32)");

            Console.WriteLine("\nНажмите любую клавишу для выхода...");

            Console.ReadKey();

        }

    }

}

## Описание кода

**Ключевые механизмы**

**1. Три подхода к вычислениям**

**Синхронная версия:**

* Блокирующие вычисления в основном потоке
* Базовый вариант для сравнения производительности

**Асинхронная версия:**

* Использование Task.Run() для выноса CPU-bound операции в пул потоков
* Не блокирует основной поток во время выполнения

**Параллельная версия:**

* Разделение работы на **4 части** с использованием Task.WhenAll()
* Параллельное вычисление частичных сумм с последующим объединением

**2. Технологии Task API**

* Task.Run() - запуск работы в пуле потоков
* Task.WhenAll() - ожидание завершения группы задач
* async/await - неблокирующее ожидание результатов
* Stopwatch - измерение производительности разных подходов

## Результат работы программы

Генерация случайных векторов...

Размер векторов: 1000000 элементов

1. Синхронное вычисление:

Результат: 24989827,275306

2. Асинхронное вычисление:

Результат: 24989827,275306

3. Параллельное вычисление (4 части):

Результат: 24989827,275305

4. Сравнение производительности:

Синхронная версия: 2 мс

Асинхронная версия: 2 мс

Параллельная версия: 0 мс

5. Проверка правильности (малые векторы):

Вектор 1: [1, 2, 3]

Вектор 2: [4, 5, 6]

Скалярное произведение: 32 (ожидалось: 32)

Нажмите любую клавишу для выхода...

# Контрольные вопросы

1. **Что такое класс Task? Чем класс Task отличается от класса Thread?**
   1. Task - высокоуровневая абстракция для асинхронных операций, часть TPL (Task Parallel Library).
   2. **Основные отличия:**
      1. Уровень абстракции: Task - высокоуровневый, Thread - низкоуровневый
      2. Пул потоков: Task автоматически использует пул, Thread создает отдельные потоки
      3. Результаты: Task<T> возвращает результат, Thread - void
      4. Координация: Task имеет встроенные механизмы ожидания (WhenAll, WhenAny)
      5. Производительность: Task эффективнее за счет переиспользования потоков
2. **Как получить идентификатор текущей задачи? Как получить идентификатор текущего потока?**
   1. int? taskId = Task.CurrentId;
   2. int threadId = Thread.CurrentThread.ManagedThreadId;
3. **Опишите возможные методы создания и запуска задач.**
   1. Task.Run
   2. Task.Factory.StartNew
   3. Конструктор + Start
   4. Async/await с возвратом Task
4. **Какие параметры задачи может установить программист при реализации задачи? Опишите возможные варианты применения параметра типа TaskCreationOptions.**
   1. **Основные параметры:**
      1. CancellationToken - для отмены задачи
      2. TaskCreationOptions - флаги создания
      3. TaskScheduler - планировщик выполнения
   2. **Варианты:**
      1. TaskCreationOptions.LongRunning // Долгая задача
      2. TaskCreationOptions.AttachedToParent // Прикрепление к родительской задаче
      3. TaskCreationOptions.HideScheduler // Отказ от наследования планировщика
      4. TaskCreationOptions.PreferFairness // Предпочтение честного планирования
      5. TaskCreationOptions.DenyChildAttach // Запрет дочерним задачам прикрепляться
      6. TaskCreationOptions.RunContinuationsAsynchronously // Асинхронное выполнение продолжений