**Асинхронное программирование**

*Асинхронное программирование* – это концепция программирования, позволяющее вернуть результат работы через какое-то время в виде асинхронного вызова.

*Асинхронный вызов* – это вызов, нарушающий обычный порядок.

Асинхронный вызов используется для оптимизации высоконагруженных приложений с частым и долгим ожиданием системы.

Асинхронность позволяет:

1. Выносить отдельные задачи из основного потока в асинхронные методы или блоки кода.
2. Выполнять операции, не блокируя вызывающий поток.
3. Запускать длительные операции без необходимости ожидания их завершения.

Языки поддержки асинхронности:

С#

JavaScript

Pyton

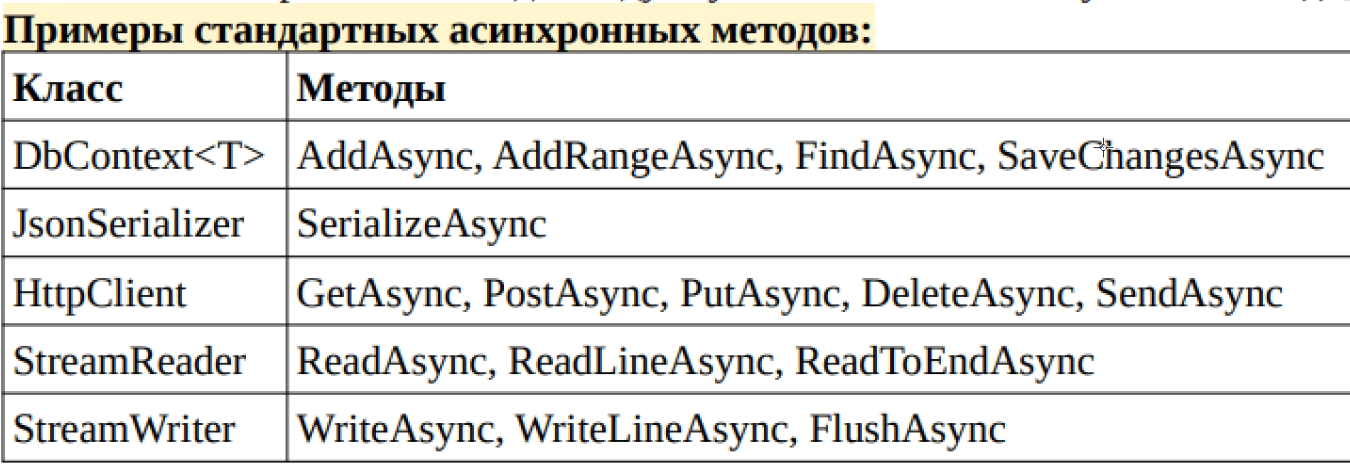
Ключевые слова C# для работы с асинхронными вызовами:

async – указывается в определении метода и означает, что в методе может быть одно или более выражений **await**

awaid –пишется перед выражением или методом, которое будет выполняться асинхронно

Признаки асинхронного метода:

1. Названия методов в формате *ИмяМетодаAsync*.

**

1. В заголовке метода (или определении) используется модификатор **async**.
2. Метод содержит одно или более выражений **await**.
3. Метод возвращает один из следующих типов:
   * void для асинхронного обработчика событий,
   * Task (задача) для асинхронного метода, который не возвращает значения. Вызов методов типа возврата Task.

*await Task.Run(() => ИмяМетода(параметры));*

или

*await ИмяМетода(параметры);*

* + Task<T> для асинхронного метода, возвращающего значения типа T

*тип переменная = await Task.Run(() => ИмяМетода(параметры));*

или

*тип переменная = await ИмяМетода(параметры);*

* + ValueTask<T>

Обзор асинхронной модели:

Требуемая модель выбирается в зависимости от того будет ли код ожидать чего-либо. Например, данных их БД, то есть задача ограниченого производительностью ввода/вывода.

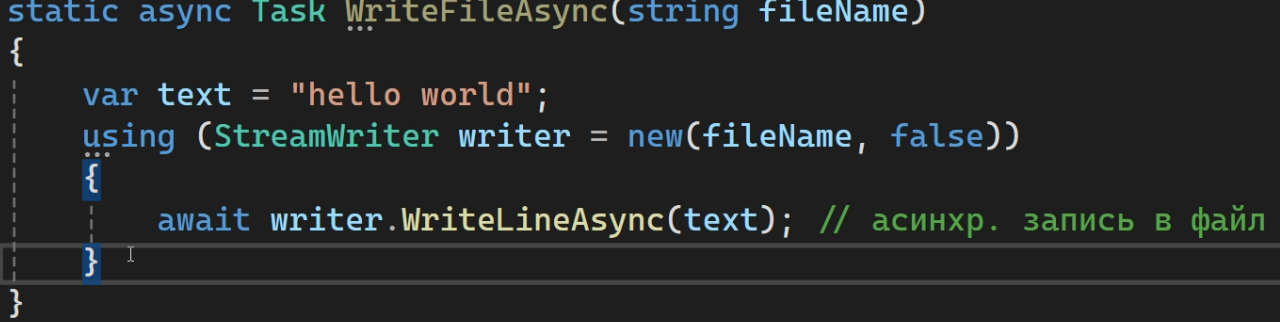
Требуемая модель выбирается в зависимости от того будет ли код выполнять сложные вычисления. То есть задача ограничена ресурсами процессора/ЦП.

1. Задача ограниченна производительностью ввода/вывода.

Решение: использовать async или await без Task.Run и выполнять await для операции, которая возвращает Task или Task<T> внутри метода async.

Вызов: *await ИмяМетодаAsync(параметры)*

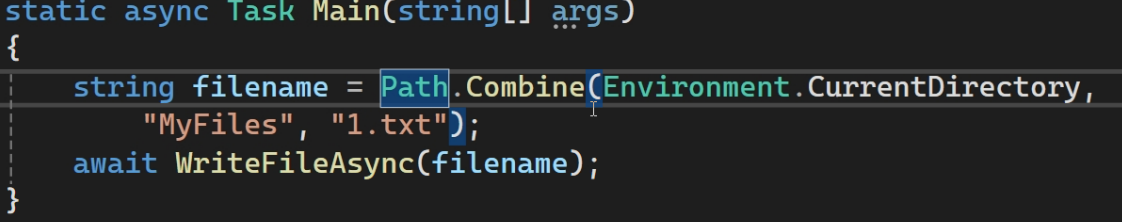
Пример асинхронного метода, записывающего строку в файл:



*using* – чтобы создавать файл и удалять его автоматически.

Составление пути к папке:

Создать папу MyFiles =>



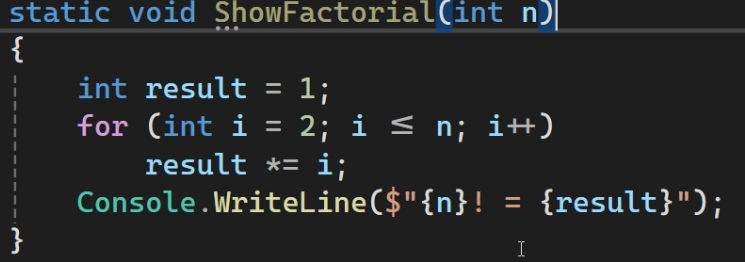
Environment.CurrentDirectory – папка текущего для экзешника

1. Задача, ограниченная ресурсами процессора.

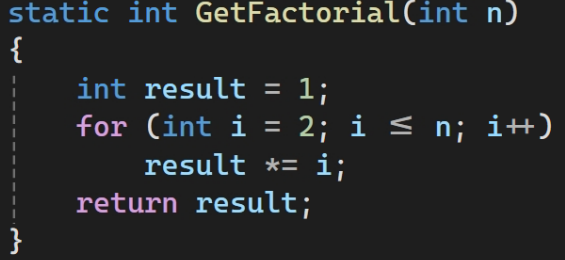
Решение: использовать async и await и перенести выполнение задачи в дополнительный поток, у которого есть Task.Run. Если к задаче применим параллелизм рассмотреть возможность использование библиотеки TPL. Выполнить await для операции, которая запускается в … потоке методом Task.Run.

*Await Task.Run(делегат Action или Func)*

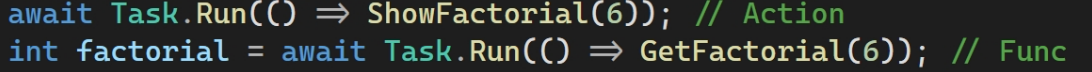
Метод выводящий факториал:



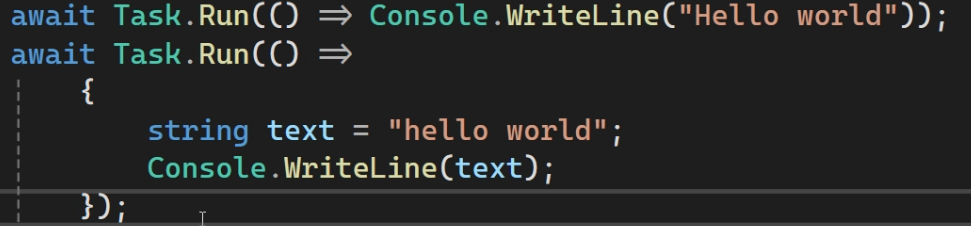
Без оптимизации и проверки.



Пример вызова:



Вместо вызова метода можно использовать лямбда-выражения:



Последовательный и параллельный вызов асинхронных операций:

1 вариант. Последовательный вызов:

*await ВызовОперации1*

*await ВызовОперации2*

*…*

Асинхронная операция начинает выполняться после завершения после завершения предыдущей асинхронной операции. Используется если результат операции зависит от результатов предыдущих операций.

2 вариант. Параллельный вызов:

*Task task1 = await ВызовОперации1*

*Task task2 = await ВызовОперации2*

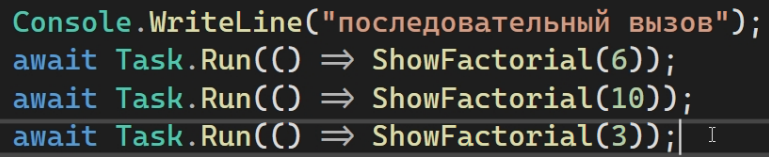
*…*

*await Task.WhenAll(список tasks);*

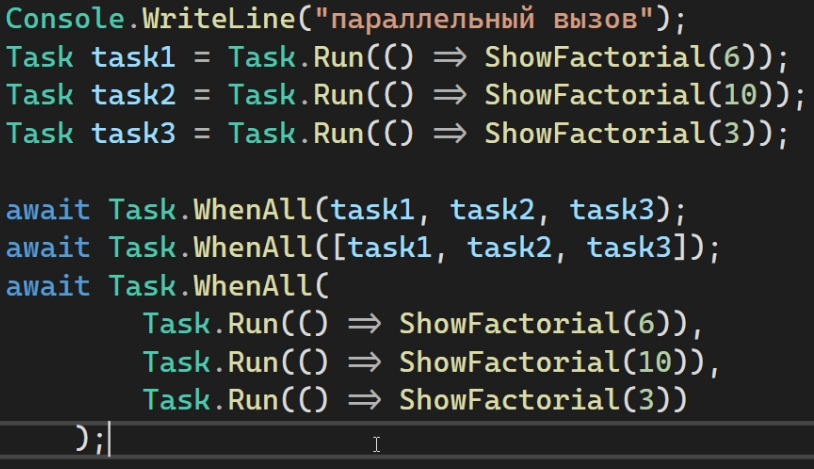
Асинхронные операции начинают выполняться одновременно.

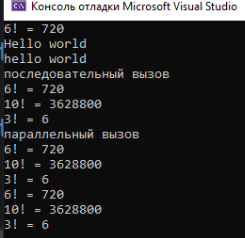
WheallAll ждет завершения всех задач и позволяет вернуть результаты.

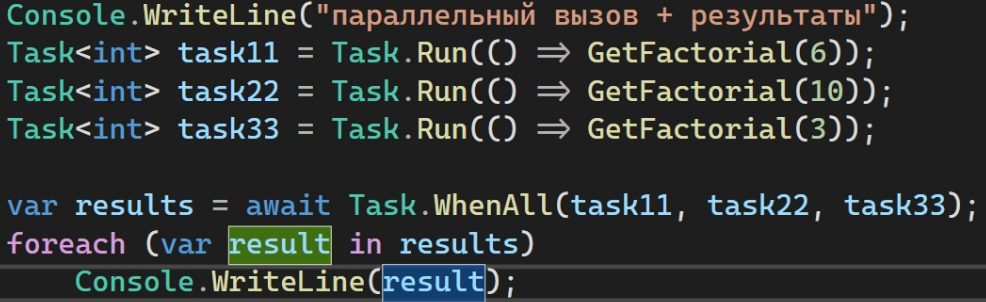
Пример последовательного вызова:

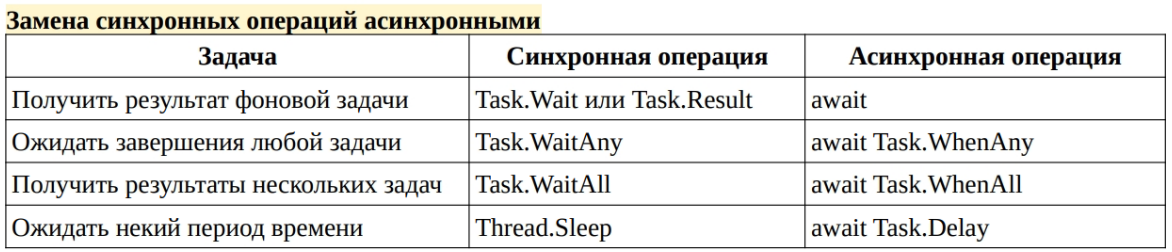


Пример параллельного вызова:

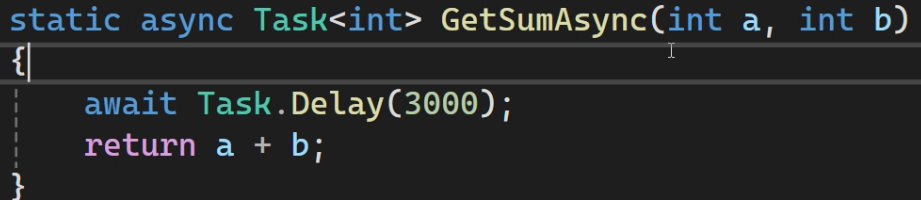




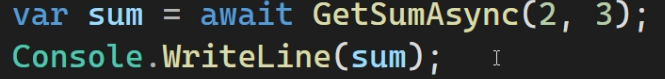




Асинхронный метод, который возвращает и имитирует задержку:



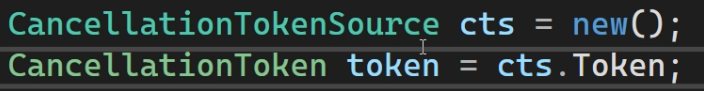
Вызов:

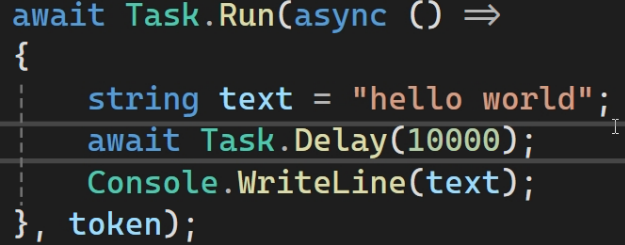


Отмена задачи. Для отмены задачи используются типы данных из пространства **System.Threading**.

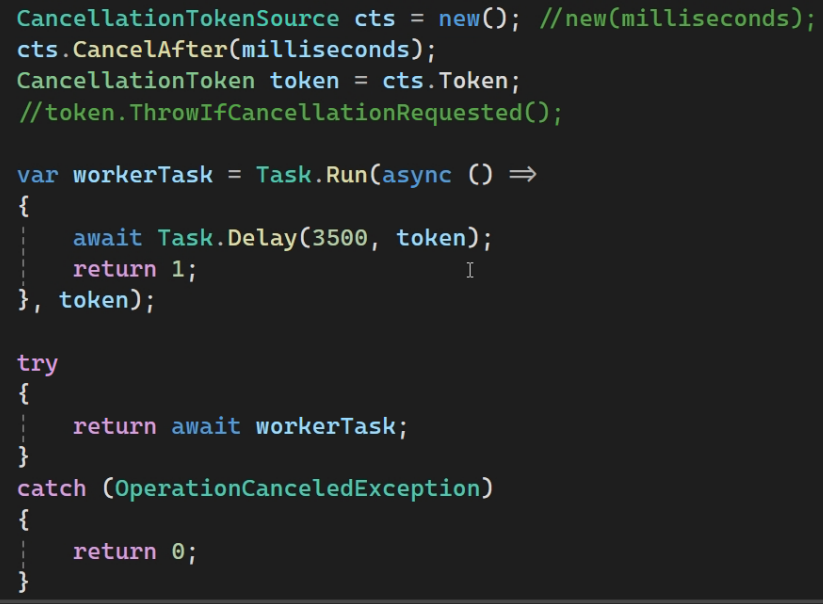
1 тип: CancellationTokenSource – это триггер отмены. Предоставляет механизм для имитирования запроса на отмену, управляет и посылает уведомление об отмене связанного **Token**.

2 тип: CancellationToken – это слушатель отмены. Предоставляет потенциальный запрос на отмену. Передается в параметрах метода, который может опрашивать его или регистрировать обратный вызов, который будет запущен при запросе отмены.





Токен передается в параметрах task1 или в конструкторе объекта класса



Отмена через 2 секунды. Не работает