Лабораторная работа № 3.  
Aсинхронное программирование

**Цель лабораторной работы**

Закрепление теоретических знаний по основам разработки асинхронных приложений.

**Постановка задачи**

Ключевыми для работы с асинхронными вызовами в C# являются два оператора: async и await, цель которых - упростить написание асинхронного кода. Они используются вместе для создания асинхронного метода.

Асинхронный метод обладает следующими **признаками**:

1. В заголовке метода используется модификатор **async**
2. Метод содержит одно или несколько выражений **await**
3. В качестве возвращаемого типа используется один из следующих:

* **void**
* **Task**
* **Task<T>**
* **ValueTask<T>**

Также стоит отметить, что слово **async**, которое указывается в определении метода, **НЕ** делает автоматически метод асинхронным. Оно лишь указывает, что данный метод может содержать одно или несколько выражений **await**.

**Пример** создания асинхронного метода.

var printTask = PrintAsync(); // 1 вызов асинхронного ме-тода

Console.WriteLine("Некоторые действия в методе Main"); // 3

await printTask;

//6 ...

// определение асинхронного метода

async Task PrintAsync()

{

// 2

Console.WriteLine("Начало метода PrintAsync");

await Task.Run(Print); // 4 Выполняется асинхронно

Console.WriteLine("Конец метода PrintAsync"); // 5

}

void Print()

{

Thread.Sleep(3000);

Console.WriteLine(

"Hello World");

}

**Пример** асинхронного вызова 3 методов.

var task1 = PrintNameAsync("Петр"); // Запуск 1 задачи

var task2 = PrintNameAsync("Владимир"); // Запуск 2 задачи

var task3 = PrintNameAsync("Сергей"); // Запуск 3 задачи

await task1; // Ожидание 1 задачи

await task2; // Ожидание 2 задачи

await task3; // Ожидание 3 задачи

// определение асинхронного метода

async Task PrintNameAsync(string name)

{

await Task.Delay(3000); // имитация продолжительной работы

Console.WriteLine(name);

}

На выполнение 3-х задач потребуется примерно 3 секунды тк все 3 задачи выполняются одновременно в разных потоках.

Определение асинхронного лямбда-выражения:

// асинхронное лямбда-выражение

Func<string, Task> printer = async (message) =>

{

await Task.Delay(1000);

Console.WriteLine(message);

};

var task1 = printer("Привет Мир!"); // 1

var task2 = printer("Конец"); // 2 Ждём выполнения 1-го мето-да и вызываем 2-й

Task.WaitAll(task1, task2); // Ждем завершения

**Получение** результата из асинхронного метода.

В качестве возвращаемого типа в асинхронном методе должны использоваться типы void, Task, Task<T> или ValueTask<T>.

**Пример с void**

PrintAsync("Привет Мир!");

PrintAsync("Привет математика!");

Console.WriteLine("Конец");

// ждем завершения задач

await Task.Delay(3000);

// определение асинхронного метода

async void PrintAsync(string message)

{

await Task.Delay(1000); // имитация продолжительной работы

Console.WriteLine(message);

}

**Пример с Task**

var task1 = PrintAsync("Привет Мир!");

var task2 = PrintAsync("Привет математика!");

await task1;

await task2;

// определение асинхронного метода

async Task PrintAsync(string message)

{

await Task.Delay(1000); // имитация продолжительной работы

Console.WriteLine(message);

}

**Пример с Task<T>**

var square5 = SquareAsync(5);

var square6 = SquareAsync(6);

Console.WriteLine("Остальные действия в методе Main");

int n1 = await square5;

int n2 = await square6;

Console.WriteLine($"n1={n1} n2={n2}"); // n1=25 n2=36

async Task<int> SquareAsync(int n)

{

await Task.Delay(3000);

var result = n \* n;

Console.WriteLine($"Квадрат числа {n} равен {result}");

return result;

}

Асинхронный метод может содержать множество выражений await. Когда система встречает в блоке кода оператор await, то выполнение в асинхронном методе останавливается, пока не завершится асинхронная задача. После завершения задачи управление переходит к следующему оператору await и так далее. Это позволяет вызывать асинхронные задачи последовательно в определенном порядке.

var task1 = PrintAsync("Привет Мир!");

var task2 = PrintAsync("Привет математика!");

var task3 = PrintAsync("Конец");

await task1;

await task2;

await task3;

.NET позволяет упростить отслеживание выполнения набора задач с помощью метода Task.WhenAll.

// определяем и запускаем задачи

var task1 = PrintAsync("Привет C#");

var task2 = PrintAsync("Привет Мир!");

var task3 = PrintAsync("Конец");

// ожидаем завершения всех задач

await Task.WhenAll(task1, task2, task3);

async Task PrintAsync(string message)

{

// имитация продолжительной операции

await Task.Delay(2000);

Console.WriteLine(message);

}

Если необходимо дождаться, когда будет выполнена хотя бы одна задача из некоторого набора задач, то применяется метод Task.WhenAny().

// определяем и запускаем задачи

var task1 = PrintAsync("Привет C#");

var task2 = PrintAsync("Привет Мир!");

var task3 = PrintAsync("Конец");

// ожидаем завершения хотя бы одной задачи

await Task.WhenAny(task1, task2, task3);

async Task PrintAsync(string message)

{

await Task.Delay(new Random().Next(1000, 2000));

// имитация продолжительной операции

Console.WriteLine(message);

}

Задачи, передаваемые в Task.WhenAll и Task.WhenAny, могут возвращать некоторое значение. В этом случае из методов Task.WhenAll и Task.WhenAny можно получить массив, который будет содержать результаты задач:

var task1 = SquareAsync(4); // определяем и запускаем задачи

var task2 = SquareAsync(5);

var task3 = SquareAsync(6);

// ожидаем завершения всех задач

int[] results = await Task.WhenAll(task1, task2, task3);

foreach (int result in results) // выводим результаты:

Console.WriteLine(result);

async Task<int> SquareAsync(int n) {

await Task.Delay(1000);

return n \* n;

}

**Задание на лабораторную работу**

1. Необходимо разработать графическое приложение для асинхронной обработки больших массивов данных.
2. Программа должна быть загружена в GitHub репозиторий.
3. Предусмотреть формы для ввода параметров расчёта и генерации данных.
4. **Для оценки «Хорошо».** Предусмотреть отображение прогресса с помощью ProgressBar.
5. **Для оценки «Отлично».** Предусмотреть отмену выполнения задач с помощью CancelationToken.
6. Защита работы включает демонстрацию работы программы.
7. Для успешной защиты:

* программа должна быть задокументирована с помощью комментариев,
* код программы не должен содержать не используемые блоки, должен быть «чистым» и стилистически верно оформлен,
* интерфейс программы должен быть гибким и удобным.

**Варианты заданий на лабораторную работу**

1. **Копирование с фильтрацией**

Асинхронно скопировать все файлы с определённым расширением (например, .txt) из одной папки в другую, выводя прогресс в консоль.

1. **Пакетное сжатие изображений**

Асинхронно сжать все .jpg-файлы в папке до 50% качества, сохраняя их в подпапку /compressed.

1. **Поиск дубликатов по размеру**

Найти все файлы с одинаковым размером (возможные дубликаты) и записать их пути в duplicates.log.

1. **Переименование файлов по шаблону**

Асинхронно переименовать все .png-файлы по шаблону image\_001.png, image\_002.png...

1. **Сбор статистики о файлах**

Посчитать общее количество файлов, их суммарный размер и записать в stats.txt.

1. **Разделение текстового файла**

Разделить большой .txt-файл на части по 100 строк каждая.

1. **Поиск слова в файлах**

Асинхронно проверить все .txt-файлы на наличие слова "error" и вывести список совпадений.

1. **Сортировка файлов по дате**

Скопировать все файлы из папки в подпапки /2023, /2024 по году создания.

1. **Конвертер текстовых кодировок**

Перекодировать все .txt-файлы из windows-1251 в UTF-8.

1. **Архивирование старых файлов**

Переместить все файлы старше 30 дней в папку /archive.

1. **Асинхронный поиск файлов по маске**

Рекурсивно найти все .log-файлы в папке и её подпапках, записать пути в файл logs.txt.

1. **Подсчёт строк в текстовых файлах**

Для всех .txt-файлов в директории посчитать количество строк и сохранить результат в формате:

1. **Конвертер CSV → JSON**

Асинхронно преобразовать все .csv-файлы в папке в JSON-формат (первая строка = ключи).

1. **Объединение файлов**

Склеить содержимое 10+ .txt-файлов в один (с разделителем "-----").

1. **Фильтрация логов**

Из всех .log-файлов выбрать только строки, содержащие "ERROR", и сохранить их в errors.log.

1. **Сортировка файлов по размеру**

Вывести список файлов в порядке убывания размера (имя → размер в МБ).

1. **Удаление временных файлов**

Рекурсивно удалить все файлы с расширением .tmp старше 1 дня.

1. **Генератор тестовых файлов**

Создать 100 .txt-файлов со случайным текстом (параллельно).

1. **Поиск самого большого файла**

Найти файл максимального размера в указанной папке и вывести его имя.

1. **Разделение файла по датам**

Разделить лог-файл server.log на отдельные файлы по датам из записей.

1. **Перекодировка видео (имитация)**

«Конвертировать» все .mp4-файлы в папке в «сжатый» формат (фактически — скопировать с новым расширением .compressed.mp4).

1. **Поиск битых изображений**

Проверить все .jpg-файлы в папке (пытаясь открыть их через Image.Load), записать «битые» в corrupted.txt.

1. **Генератор MD5-хешей**

Для каждого файла в папке вычислить хеш и сохранить в новой папке, в качестве имени выбрать имя исходного файла, содержимое – хеш.

1. **Фильтр по разрешению изображений**

Скопировать все .png-файлы с шириной > 1000 пикселей в папку /large.

1. **Сборщик мусора**

Удалить все пустые файлы (0 байт) и пустые папки в указанной директории.

1. **Переименование по шаблону**

Заменить все пробелы в именах файлов на \_ (например, "my file.txt" → "my\_file.txt").

1. **Анализатор частоты слов**

Для всех .json-файлов посчитать топ-20 самых частых слов по заданному ключу (игнорируя предлоги).

1. **Синхронизация папок**

Скопировать из FolderA в FolderB только файлы, которых нет в FolderB или которые новее.

1. **Поиск дубликатов по содержимому**

Найти файлы с одинаковым содержимым (через хеширование) и записать группы дубликатов в duplicates.txt.

1. **Ротация логов**

Оставить только 5 самых свежих .log-файлов, остальные удалить.

**Пример разработки приложения**

**Задание**

Для всех .txt-файлов посчитать топ-10 самых частых слов (игнорируя предлоги).

**1. Подготовка проекта**

**Что делаем:**

* Создаем Windows Forms приложение в Visual Studio 2022
* Добавляем два класса: IgnoreWords и WordFrequencyAnalyzer

**Почему именно так:**

* Отдельный класс IgnoreWords обеспечивает легкое изменение списка игнорируемых слов
* Основная логика инкапсулирована в WordFrequencyAnalyzer для удобства повторного использования

**2. Класс IgnoreWords**

namespace ConsoleApp1;

public static class IgnoreWords

{

public static HashSet<string> Default => new()

{

"в", "на", "с", "по", "к", "из", "у", "за", "от", "для", "о", "без", "до", "под", "над", "при", "про"

};

}

**Пояснения:**

* HashSet используется для быстрого поиска (O(1) при проверке игнорируемых слов
* static класс, так как не требует состояния и содержит только константные данные
* Метод Default возвращает новый экземпляр при каждом обращении, что безопасно для многопоточности

3. Класс WordFrequencyAnalyzer

3.1. Поля класса

private HashSet<string> ignoredWords;

private ConcurrentDictionary<string, int> allWordCounts;

public IEqualityComparer<string>? Comparer { get; }

**Пояснения:**

**ConcurrentDictionary**:

* Гарантирует потокобезопасность при параллельной обработке файлов
* Автоматически обрабатывает конкурирующие запросы на обновление счетчиков

**StringComparer.OrdinalIgnoreCase**:

* Обеспечивает сравнение строк без учета регистра
* Более эффективен, чем ToLower() при каждом сравнении

**HashSet для ignoredWords**:

* Оптимален для частых операций поиска
* Потребляет немного больше памяти, но дает значительный выигрыш в скорости

3.2. Конструктор

public WordFrequencyAnalyzer()

{

allWordCounts = new ConcurrentDictionary<string, int>();

Comparer = StringComparer.OrdinalIgnoreCase;

ignoredWords = IgnoreWords.Default;

}

**Особенности:**

* Инициализация коллекций в конструкторе предотвращает NullReferenceException
* Использование StringComparer.OrdinalIgnoreCase по умолчанию

3.3. Методы настройки (Fluent Interface)

public WordFrequencyAnalyzer AddComparer(IEqualityComparer<string> comparer)

{

allWordCounts = new ConcurrentDictionary<string, int>(comparer);

return this;

}

public WordFrequencyAnalyzer AddIgnoreWords(HashSet<string> ignoredWords)

{

this.ignoredWords = ignoredWords;

return this;

}

**Почему fluent interface:**

* Позволяет писать цепочки вызовов:

new Analyzer().AddComparer(...).AddIgnoreWords(...)

* Делает код более читаемым и выразительным

3.4. Основной метод анализа

public async Task AnalyzeDirectoryAsync(string directoryPath)

{

var txtFiles = Directory.GetFiles(directoryPath, "\*.txt");

allWordCounts.Clear();

var processingTasks = txtFiles.Select(file => ProcessFileAsync(file, allWordCounts));

await Task.WhenAll(processingTasks);

}

**Ключевые моменты:**

* Directory.GetFiles возвращает все .txt файлы в указанной папке
* Clear() очищает результаты предыдущих анализов
* Task.WhenAll запускает параллельную обработку файлов

3.5. Обработка файла

private async Task ProcessFileAsync(string filePath, ConcurrentDictionary<string, int> wordCounts)

{

var content = await File.ReadAllTextAsync(filePath);

var words = ExtractWords(content);

foreach (var word in words)

{

if (wordCounts.ContainsKey(word))

{

wordCounts[word]++;

}

else

{

wordCounts[word] = 1;

}

}

}

**Оптимизации:**

* AddOrUpdate — атомарная операция, безопасная для многопоточности
* Асинхронное чтение (ReadAllTextAsync) не блокирует основной поток

3.6. Извлечение слов

private IEnumerable<string> ExtractWords(string text)

{

var matches = Regex.Matches(text, @"\b[\p{L}-]+\b");

return matches

.Select(m => m.Value.ToLower())

.Where(w => !ignoredWords.Contains(w));

}

**Детали реализации:**

* Регулярное выражение:

\b — границы слова

[\p{L}-] — буквы любого алфавита и дефис

+ — одно или более вхождений

* Преобразование в нижний регистр (ToLower()) для нормализации
* Фильтрация игнорируемых слов через HashSet (быстрый поиск)

3.7. Получение результатов

public List<KeyValuePair<string, int>> GetTopWords(int count)

{

return allWordCounts

.OrderByDescending(pair => pair.Value)

.Take(count)

.ToList();

}

**Особенности:**

* OrderByDescending сортирует по убыванию частоты
* Take ограничивает количество результатов
* Преобразование в List для удобства использования

4. Пример использования

private static List<KeyValuePair<string, int>> GetTopWords(IDictionary<string, int> wordCounts, int count)

{

return wordCounts

.OrderByDescending(pair => pair.Value)

.Take(count)

.ToList();

}

public List<KeyValuePair<string, int>> GetTopWords(int count)

{

return GetTopWords(allWordCounts, count);

}

**Пошаговая работа:**

* Создаем анализатор с настройками
* Запускаем анализ папки (асинхронно)
* Получаем топ-10 слов

Код целиком:

using ConsoleApp1;

var wfa = new WordFrequencyAnalyzer()

.AddComparer(StringComparer.OrdinalIgnoreCase)

.AddIgnoreWords(IgnoreWords.Default);

await wfa.AnalyzeDirectoryAsync("C:\\Рабочий стол\\1");

var words = wfa.GetTopWords(10);

Console.ReadKey();

namespace ConsoleApp1;

public static class IgnoreWords

{

public static HashSet<string> Default => new()

{

"в", "на", "с", "по", "к", "из", "у", "за", "от", "для", "о", "без", "до", "под", "над", "при", "про"

};

}

using System.Collections.Concurrent;

using System.Text.RegularExpressions;

namespace ConsoleApp1;

public class WordFrequencyAnalyzer

{

private HashSet<string> ignoredWords;

private ConcurrentDictionary<string, int> allWordCounts;

public IEqualityComparer<string>? Comparer { get; }

public WordFrequencyAnalyzer()

{

allWordCounts = new ConcurrentDictionary<string, int>();

Comparer = StringComparer.OrdinalIgnoreCase;

ignoredWords = IgnoreWords.Default;

}

public WordFrequencyAnalyzer AddComparer(IEqualityComparer<string> comparer)

{

allWordCounts = new ConcurrentDictionary<string, int>(comparer);

return this;

}

public WordFrequencyAnalyzer AddIgnoreWords(HashSet<string> ignoredWords)

{

this.ignoredWords = ignoredWords;

return this;

}

public async Task AnalyzeDirectoryAsync(string directoryPath)

{

var txtFiles = Directory.GetFiles(directoryPath, "\*.txt");

allWordCounts.Clear();

var processingTasks = txtFiles.Select(file => ProcessFileAsync(file, allWordCounts));

await Task.WhenAll(processingTasks);

}

private async Task ProcessFileAsync(string filePath, ConcurrentDictionary<string, int> wordCounts)

{

var content = await File.ReadAllTextAsync(filePath);

var words = ExtractWords(content);

foreach (var word in words)

{

if (wordCounts.ContainsKey(word))

{

wordCounts[word]++;

}

else

{

wordCounts[word] = 1;

}

}

}

private IEnumerable<string> ExtractWords(string text)

{

var matches = Regex.Matches(text, @"\b[\p{L}-]+\b");

return matches

.Select(m => m.Value.ToLower())

.Where(w => !ignoredWords.Contains(w));

}

private static List<KeyValuePair<string, int>> GetTopWords(IDictionary<string, int> wordCounts, int count)

{

return wordCounts

.OrderByDescending(pair => pair.Value)

.Take(count)

.ToList();

}

public List<KeyValuePair<string, int>> GetTopWords(int count)

{

return GetTopWords(allWordCounts, count);

}

}

# Содержание пояснительной записки

1. Постановка задачи. Приводится теоретический материал, использованный при написании приложения.

2. Формулировка задания и вариант. Приводится задание на лабораторную работу и вариант этого задания.

3. Описание выполняемых действий. Необходимо привести описание последовательности разработки программы, реализации используемых методов, алгоритмов, блок-схем.

4. Анализ результатов. Привести анализ входных и выходных данных. Показать результаты выполнения программного кода. Предоставить скриншоты обработки тестовых примеров. Сделать выводы.

5. Листинг программы. Привести листинг разработанного программного кода, содержание файлов входных и выходных данных.

# Используемое программное обеспечение

1. Среда программирования MS Visual Studio Community 2022 (Свободно распространяемое программное обеспечение (в учебных целях));
2. Microsoft Office Standard 2007 (Open License: 42267924);
3. Open Office (Свободно распространяемое программное обеспечение).
4. Браузер (Свободно распространяемое программное обеспечение).

# Список литературы

* + - 1. Мейер Б. Объектно-ориентированное программирование и программная инженерия [Электронный ресурс]/ Мейер Б. – Электрон. текстовые данные. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. – 285 c.
      2. Биллиг, В. A. Основы объектного программирования на С# (C# 3.0, Visual Studio 2008) [Электронный ресурс] : учебное пособие / В. A. Биллиг. — Электрон. текстовые данные. — Москва, Саратов : Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2017. — 583 c. — 978-5-4487-0145-0. — Режим доступа: http://www.iprbookshop.ru/72339.html
      3. Павловская, Т. А. Программирование на языке высокого уровня C# [Электронный ресурс] / Т. А. Павловская. — 2-е изд. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 245 c. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/73713.html
      4. Агапов, В. П. Основы программирования на языке С# [Электронный ресурс] : учебное пособие / В. П. Агапов. — Электрон. текстовые данные. — М. : Московский государственный строительный университет, ЭБС АСВ, 2012. — 128 c. — 978-5-7264-0576-6. — Режим доступа: http://www.iprbookshop.ru/16366.html
      5. Медведев, М. А. Программирование на СИ# [Электронный ресурс] : учебное пособие / М. А. Медведев, А. Н. Медведев ; под ред. А. В. Присяжный. — Электрон. текстовые данные. — Екатеринбург : Уральский федеральный университет, ЭБС АСВ, 2015. — 64 c. — 978-5-7996-1561-1. — Режим доступа: http://www.iprbookshop.ru/69667.html
      6. Казанский А.А. Объектно-ориентированное программирование на языке Microsoft Visual С# в среде разработки Microsoft Visual Studio 2008 и .NET Framework. 4.3 [Электронный ресурс]: учебное пособие и практикум/ Казанский А.А.— Электрон. текстовые данные.— М.: Московский государственный строительный университет, ЭБС АСВ, 2011.— 180 c
      7. Уйманова Н.А. Основы объектно-ориентированного программирования [Электронный ресурс]: практикум/ Уйманова Н.А., Таспаева М.Г.— Электрон. текстовые данные.— Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2017.— 156 c.— Режим доступа: http://www.iprbookshop.ru/78808.html.— ЭБС «IPRbooks»
      8. Новиков П.В. Объектно-ориентированное программирование [Электронный ресурс]: учебно-методическое пособие к лабораторным работам/ Новиков П.В.— Электрон. текстовые данные.— Саратов: Вузовское образование, 2017.— 124 c.— Режим доступа: http://www.iprbookshop.ru/64650.html.— ЭБС «IPRbooks»