МИНИCTEPCTBO НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии

Департамент цифровых, робототехнических систем и электроники

Отчет к практической работе № 15

Тема: **«Ожидание завершения асинхронного метода с использованием тайм-аута»**

Дисциплина: **«Алгоритмы и структуры данных»**

|  |  |  |
| --- | --- | --- |
|  |  | **Выполнил:** |
|  |  | Студент группы ПИЖ-б-о-23-1, направление подготовки: 09.03.04 «Программная инженерия»  Панчешный Александр Алексеевич  **Проверил:**  Доцент ДЦРСИЭ  Николаев Евгений Иванович |

Ставрополь 2025

**Цель лабораторной работы:**

Научиться использовать механизм ожидания завершения работы асинхронного метода с применением типа **IAsyncResult** и тайм-аута.

**Задачи лабораторной работы:**

1. Освоить использование механизма тайм-аутов при асинхронных операциях.
2. Научиться выводить информацию о ходе выполнения асинхронного метода.
3. Приобрести навыки отслеживания выполнения асинхронного метода.

**Ход работы**

**Индивидуальное задание:**

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Исходный код:**

using System;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

class Program

{

*// Библиотечный делегат Func для нашей задачи*

static Func<int, int, int[]> filterNumbersDelegate = FilterNumbers;

static async Task Main(string[] args)

{

Console.WriteLine("Программа фильтрации чисел запущена.");

*// Параметры для метода*

Console.Write("Введите размер массива: ");

int size = int.Parse(Console.ReadLine());

Console.Write("Введите число для сравнения: ");

int number = int.Parse(Console.ReadLine());

*// Запуск асинхронного выполнения с тайм-аутом*

Console.WriteLine("\nЗапуск асинхронной операции...");

var cts = new CancellationTokenSource();

var task = Task.Run(() => filterNumbersDelegate(size, number, cts.Token), cts.Token);

*// Установка тайм-аута в 5 секунд*

cts.CancelAfter(5000);

try

{

*// Мониторинг процесса выполнения с выводом информации*

while (!task.IsCompleted)

{

if (cts.Token.IsCancellationRequested)

{

Console.WriteLine("\nПревышено время ожидания (тайм-аут)!");

break;

}

Console.Write(".");

await Task.Delay(500);

}

if (task.IsCompletedSuccessfully)

{

Console.WriteLine("\n\nОперация завершена успешно!");

*// Получение результата*

int[] result = await task;

*// Вывод результатов*

Console.WriteLine($"\nИсходный массив: [{string.Join(", ", GenerateRandomArray(size))}]");

Console.WriteLine($"Число для сравнения: {number}");

Console.WriteLine($"Результат (числа, отличающиеся не более чем на 4): [{string.Join(", ", result)}]");

}

}

catch (OperationCanceledException)

{

Console.WriteLine("\nОперация отменена по тайм-ауту!");

}

catch (Exception ex)

{

Console.WriteLine($"\nПроизошла ошибка: {ex.Message}");

}

}

*// Модифицированный метод с поддержкой отмены и выводом прогресса*

static int[] FilterNumbers(int size, int number, CancellationToken token)

{

Console.WriteLine("\nНачало обработки массива...");

var randomArray = GenerateRandomArray(size);

var result = new System.Collections.Generic.List<int>();

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

{

*// Проверка отмены операции*

if (token.IsCancellationRequested)

{

Console.WriteLine($"\nОбработано {i} из {randomArray.Length} элементов перед отменой");

token.ThrowIfCancellationRequested();

}

*// Имитация обработки каждого элемента*

Thread.Sleep(300);

if (Math.Abs(randomArray[i] - number) <= 4)

{

result.Add(randomArray[i]);

}

*// Вывод прогресса каждые 5 элементов*

if (i % 5 == 0)

{

Console.WriteLine($"Обработано {i + 1} из {randomArray.Length} элементов...");

}

}

Console.WriteLine("Обработка массива завершена!");

return result.ToArray();

}

*// Генерация массива случайных чисел*

static int[] GenerateRandomArray(int size)

{

Random rand = new Random();

int[] array = new int[size];

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

{

array[i] = rand.Next(0, 100); *// числа от 0 до 100*

}

return array;

}

}

**Контрольные вопросы**

**1. Для чего применяется тип IAsyncResult?**

**IAsyncResult** — это интерфейс, используемый в **шаблоне APM** (Asynchronous Programming Model) для управления асинхронными операциями. Его основные функции:

* **Мониторинг состояния**: Свойство IsCompleted позволяет проверить, завершена ли операция.
* **Ожидание завершения**: Через AsyncWaitHandle (возвращает WaitHandle) можно блокировать поток до завершения операции.
* **Передача контекста**: Свойство AsyncState хранит пользовательские данные (например, объект состояния).

**Пример**:

Func<int, string> asyncMethod = SomeLongRunningTask;

IAsyncResult result = asyncMethod.BeginInvoke(42, null, null);

// Ожидание завершения с тайм-аутом (2 секунды)

if (result.AsyncWaitHandle.WaitOne(2000))

{

string resultData = asyncMethod.EndInvoke(result);

Console.WriteLine(resultData);

}

else

{

Console.WriteLine("Тайм-аут операции!");

}

**Современная альтернатива**: Вместо IAsyncResult рекомендуется использовать Task и async/await.

**2. Как реализовать ожидание завершения выполнения асинхронного метода с использованием тайм-аута?**

Для реализации ожидания с тайм-аутом можно использовать:

1. WaitHandle (для IAsyncResult)

IAsyncResult result = delegate.BeginInvoke(null, null);

bool completed = result.AsyncWaitHandle.WaitOne(3000); // Тайм-аут 3 секунды

if (completed)

{

var data = delegate.EndInvoke(result);

}

else

{

Console.WriteLine("Тайм-аут!");

}

2. CancellationTokenSource (для Task)

var cts = new CancellationTokenSource(3000); // Тайм-аут 3 секунды

try

{

await Task.Run(() => SomeMethod(), cts.Token);

}

catch (TaskCanceledException)

{

Console.WriteLine("Тайм-аут операции!");

}

3. Task.Wait(timeout)

Task task = Task.Run(() => SomeMethod());

if (!task.Wait(3000)) // Блокирует поток на 3 секунды

{

Console.WriteLine("Тайм-аут!");

}

**Разница**:

* WaitHandle — для старых делегатов (APM).
* CancellationTokenSource — современный способ с отменой.
* Task.Wait() — блокирует поток (не рекомендуется в UI).

**3. Поясните назначение метода WaitOne( )**

**WaitOne()** — метод класса WaitHandle (например, у Mutex, Semaphore, или AsyncWaitHandle из IAsyncResult). Его функции:

* **Блокирует текущий поток** до получения сигнала от WaitHandle.
* **Возвращает**true, если сигнал получен, и false при тайм-ауте.
* **Поддержка тайм-аута**: Можно указать время ожидания (в миллисекундах).

**Примеры использования**:

1. С тайм-аутом:

IAsyncResult result = delegate.BeginInvoke(null, null);

if (!result.AsyncWaitHandle.WaitOne(2000)) // Ждем 2 секунды

{

Console.WriteLine("Операция не завершена вовремя!");

}

2. Без тайм-аута (бесконечное ожидание):

result.AsyncWaitHandle.WaitOne(); // Блокирует поток до завершения

**Важно**:

* В современных приложениях вместо WaitOne() используют await с Task.
* WaitOne() блокирует поток — не подходит для UI-приложений.

**Альтернатива**: Для асинхронного ожидания в .NET 4.5+ можно использовать WaitHandle.WaitOneAsync() (из пакета System.Threading.Channels).