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 № 14

Тема: **«Применение асинхронных делегатов для реализации многопоточности»**

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 работы:** научиться использовать делегаты для организации многопоточного приложения.

**Задачи работы:**

* Научиться объявлять, инициализировать и запускать потоки с использованием пользовательских делегатов
* Научиться запускать потоки с использованием библиотечных делегатов в Action<T> и Func<T>
* Научиться запускать параллельные потоки с использованием лямбда-выражений

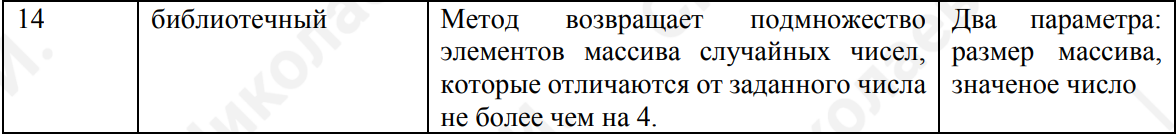
**Ход работы**

**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.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*

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

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

*// Мониторинг процесса выполнения*

while (!task.IsCompleted)

{

Console.Write(".");

await Task.Delay(500);

}

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

*// 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)}]");

}

*// Метод, который фильтрует числа*

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

{

*// Имитация длительной операции*

Task.Delay(2000).Wait();

var randomArray = GenerateRandomArray(size);

return randomArray.Where(x => Math.Abs(x - number) <= 4).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. Поясните назначение типа IAsyncResult.**

**IAsyncResult** — это интерфейс, используемый для мониторинга и управления асинхронными операциями. Он предоставляет:

* **IsCompleted**: проверяет завершение операции.
* **AsyncWaitHandle**: позволяет ожидать завершения через WaitHandle.
* **AsyncState**: хранит пользовательские данные, переданные при запуске операции.

Пример:

Func<int, int> делегат = SomeMethod;

делегат.BeginInvoke(5, null, null); // Возвращает IAsyncResult

Используется в старом шаблоне APM (Asynchronous Programming Model), но в современных приложениях предпочтительны Task и async/await.

**2. Для чего используется метод Thread.Sleep()?**

Thread.Sleep() приостанавливает текущий поток на указанное время (в миллисекундах). Используется для:

* Имитации долгой операции.
* Синхронизации (например, в тестах).

**Важно:**

* Блокирует поток, что нежелательно в UI-приложениях.
* В асинхронном коде лучше использовать await Task.Delay(), который не блокирует поток.

**Пример:**

Thread.Sleep(1000); // Приостановка на 1 секунду

**3. Поясните механизм возврата значения из метода асинхронного делегата.**

Для возврата значения из асинхронного делегата:

1. Использование EndInvoke (для старых делегатов):

Func<int, string> делегат = SomeMethod;

IAsyncResult result = делегат.BeginInvoke(5, null, null);

string результат = делегат.EndInvoke(result); // Блокирует поток до завершения

2. Использование Task (современный способ):

Task<string> task = Task.Run(() => SomeMethod(5));

string результат = await task; // Не блокирует поток

**4. Как произвести возврат более одного значения из метода?**

Способы:

1. Кортежи (Tuple):

(int, string) GetData() => (42, "Ответ");

var (число, текст) = GetData();

2. Класс/структура:

class Result { public int Number; public string Text; }

Result GetData() => new Result { Number = 42, Text = "Ответ" };

3. out-параметры:

void GetData(out int число, out string текст) { число = 42; текст = "Ответ"; }

GetData(out var num, out var text);

**5. Какая разница существует между библиотечными делегатами, пользовательскими типами делегатов и лямбда-выражениями? Являются ли эти делегаты взаимозаменяемыми при реализации асинхронного вызова методов?**

* Библиотечные делегаты (Func<T>, Action, Predicate):
  + Стандартные обобщенные типы.
  + Удобны для большинства сценариев.
* Пользовательские делегаты:
  + Определяются вручную (delegate int MyDelegate(string s);).
  + Используются для специфичных сигнатур или лучшей читаемости.
* Лямбда-выражения:
  + Анонимные методы, которые можно присвоить делегатам.
  + Пример: Func<int, int> square = x => x \* x;.

Взаимозаменяемость:

* Все три варианта можно использовать для асинхронных вызовов через Task.Run() или BeginInvoke/EndInvoke.
* Лямбды часто комбинируются с библиотечными делегатами:

Task.Run(() => Console.WriteLine("Лямбда с Func"));

* Пользовательские делегаты требуют явного определения, но работают аналогично.

**Итог**:

* Предпочтительны Func/Action + лямбды для краткости.
* Пользовательские делегаты — для сложных сигнатур или документирования намерений.

Пример взаимозаменяемости:

// Библиотечный делегат + лямбда

Func<int, bool> func = x => x > 0;

Task.Run(() => func(5));

// Пользовательский делегат

delegate bool MyPredicate(int x);

MyPredicate pred = x => x > 0;

Task.Run(() => pred(5));