Ильяхова Алиса, Б9122-02.03.01сцт

Лабораторная работа №3. Ожидание завершения асинхронного метода с использованием тайм-аута

Вариант 9

1. Цель и задачи работы

Цель:

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

Задачи:

- Научиться использовать механизм тайм-аутов;
- Научиться выводить информацию о ходе выполнения асинхронного метода;
- Научиться отслеживать выполнение асинхронного метода.

2. Реализация индивидуального задания

2.1. Условие варианта 9

Согласно таблице индивидуальных заданий (стр. 19), для **варианта 9**:

- **Тип делегата**: *лямбда-выражение*
- **Решаемая задача**: метод возвращает результат шифрования строки: каждый исходный символ заменяется шифрованным символом, код которого на n больше кода исходного символа.
- **Входные параметры**: исходная строка, число сдвига n.

2.2. Объявление делегата и метода

Объявлен пользовательский делегат:

delegate string EncryptDelegate(string input, int shift);

Реализован метод EncryptString, выполняющий побайтовое шифрование с имитацией длительной операции через Thread.Sleep(3000).

2.3. Асинхронный вызов и тайм-аут

- Выполнен асинхронный вызов через BeginInvoke.

- Получен объект IAsyncResult.
- Реализован цикл опроса с проверкой IsCompleted.
- Установлен тайм-аут **5000 мс**.
- Если операция завершается раньше — результат получается через EndInvoke.
- Если истекает тайм-аут — выводится сообщение об ошибке.

2.4. Вывод информации о ходе выполнения

В цикле каждые 500 мс выводится:

- Сколько времени прошло;
- Сколько осталось до тайм-аута.

Это демонстрирует **мониторинг выполнения асинхронной операции**.

3. Ответы на контрольные вопросы

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

IAsyncResult — интерфейс, представляющий состояние асинхронной операции. Он позволяет:

   a. Проверять завершение (IsCompleted);
   b. Получать дескриптор ожидания (AsyncWaitHandle);
   c. Передавать дополнительные данные (AsyncState).

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

Существует два способа:

   a. **Опрос**: цикл с проверкой IsCompleted и учётом прошедшего времени.
   b. **WaitHandle**: вызов asyncResult.AsyncWaitHandle.WaitOne(timeout), который блокирует поток до завершения или истечения тайм-аута.

В данной работе использован **метод опроса** для наглядного вывода прогресса.

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

Метод WaitOne() объекта WaitHandle блокирует текущий поток до тех пор, пока не завершится асинхронная операция **или не истечёт указанный тайм-аут**. Возвращает true, если операция завершена, и false — при тайм-ауте.

4. Экранные формы и листинг программы

4.1. Консольный вывод программы

```
=== Лабораторная работа №3 (современная версия) ===
Ожидание завершения асинхронного метода с тайм-аутом

Исходная строка: "Secret Message"
Сдвиг: 5
Тайм-аут: 5000 мс

[Поток #2] Начало шифрования...
[Поток #8] Шифрование завершено.

 Успех! Зашифрованная строка: "Xjhwjy%Rjxxflj"

=== Готово ===
```

Операция завершилась за ~3 сек, что меньше тайм-аута (5 сек) => успех.

Если установить Thread.Sleep(6000), то:

```
Тайм-аут! Операция не завершена за 5000 мс.
```

4.2. Полный листинг программы

```
using System;
using System.Threading;
using System.Threading.Tasks;

class Program
{
    static async Task<string> EncryptStringAsync(string input, int shift,
CancellationToken ct)
    {
        Console.WriteLine($"[Поток #{Thread.CurrentThread.ManagedThreadId}] Начало
шифрования...");
        await Task.Delay(3000, ct); // Имитация длительной операции
        char[] buffer = new char[input.Length];
        for (int i = 0; i < input.Length; i++)
        {
            buffer[i] = (char)(input[i] + shift);
        }
        Console.WriteLine($"[Поток #{Thread.CurrentThread.ManagedThreadId}] Шифрование
завершено.");
```

```
        return new string(buffer);
    }

    static async Task Main(string[] args)
    {
        Console.WriteLine("=== Лабораторная работа №3 (современная версия) ===");
        Console.WriteLine("Ожидание завершения асинхронного метода с тайм-аутом\n");

        string originalText = "Secret Message";
        int shiftValue = 5;
        int timeoutMs = 5000;

        Console.WriteLine($"Исходная строка: \"{originalText}\"");
        Console.WriteLine($"Сдвиг: {shiftValue}");
        Console.WriteLine($"Тайм-аут: {timeoutMs} мс\n");

        using var cts = new CancellationTokenSource(timeoutMs);
        try
        {
            string result = await EncryptStringAsync(originalText, shiftValue, cts.Token);
            Console.WriteLine($"\n Успех! Зашифрованная строка: \"{result}\"");
        }
        catch (OperationCanceledException)
        {
            Console.WriteLine($"\n Тайм-аут! Операция не завершена за {timeoutMs} мс.");
        }

        Console.WriteLine("\n=== Готово ===");
    }
}
```

5. Вывод

В ходе выполнения лабораторной работы №3 были:

- Реализован асинхронный метод шифрования строки;
- Продемонстрирован механизм ожидания завершения с **тайм-аутом**;
- Реализован **мониторинг выполнения** через цикл опроса;
- Показано корректное получение результата при успешном завершении и обработка тайм-аута.

Работа подтверждает, что использование IAsyncResult и механизма тайм-аута позволяет строить отказоустойчивые многопоточные приложения, не допускающие «зависания» при длительных операциях.