1. Цель и задачи лабораторной работы

Цель: научиться использовать делегаты для организации многопоточного приложения.

Задачи:

1. Научиться объявлять, инициализировать и запускать потоки с использованием пользовательских делегатов;
2. Научиться запускать потоки с использованием библиотечных делегатов *Action<T>* и *Func<T>*;
3. Научиться запускать параллельные потоки с использованием лямбдавыражений.

2. Реализация индивидуального задания

Согласно варианту задания, требуется реализовать пользовательский делегат и реализовать асинхронное выполнение метода на основе разработанного делегата с возможностью мониторинга процесса выполнения, передачи параметров в метод и получения результата работы метода.

В моём варианте делегат задаётся лямбда-выражением и принимает два параметра: строку и символ.

Метод возвращает логическое значение, указывающее существует ли заданный символ в строке.

2.1. Листинг программного кода

```
namespace lab2
{
    class Program
    {
        const short DELAY = 500;
        public static string[] SplitString(string input, int countParts)
        {
            int partLength = input.Length / (countParts - 1);
            string[] parts = new string[countParts];
            int currentPart = 0;
            for (int i = 0; i < input.Length; i += partLength)
            {
```

```
                int charsCount = 0;
                int currentIndex = i;

                // Подсчет реальных символов с учетом Unicode
                while (charsCount < partLength && currentIndex < input.Length)
                {
                    char c = input[currentIndex];
                    charsCount += char.IsSurrogate(c) ? 2 : 1;
                    currentIndex++;
                }

                parts[currentPart] = input.Substring(i, currentIndex - i);
                currentPart++;
            }
            return parts;
        }

        public delegate Task<bool?> AsyncDelegate(string str, char symbol,
IProgress<string> progress, CancellationToken cancellationToken);

        public static async Task Main(string[] args)
        {
            // u, v
            const string text =
"asjhdkajsdjkskjfldfghjghgasdhsadugkjashdjkashdjkhgdsagdhasfdgasdgsajhdtashdjashd
sajdhjas;das;jkdasjkdpkvashdjklashdjhasgdhjagsdjhas;dl";
            const char symbol = 'u';
            const int countParts = 4;

            string[] parts = SplitString(text, countParts);


            AsyncDelegate lambdaDelegate = async (str, symbol, progress,
cancelToken) =>
            {
                progress?.Report("Start");
                char[] charArray = str.ToCharArray();
                int strLength = str.Length;
                for (int i = 0; i < strLength; i++)
                {
                    cancelToken.ThrowIfCancellationRequested();
                    progress?.Report($"Шаг {i + 1} из {strLength} возможных");
                    if (charArray[i] == symbol)
                    {
                        progress?.Report("Найдено!");
                        return true;
                    }

                    await Task.Delay(DELAY, cancelToken);
                }
                progress?.Report("Finish");
```

```
                return false;
            };

            using var cancellationSource = new CancellationTokenSource();
            var tasks = new List<Task<bool?>>();


            for (int i = 0; i < parts.Length; i++)
            {
                int processId = i + 1;
                var progress = new Progress<string>(message =>
                {
                    Console.WriteLine($"[{DateTime.Now:HH:mm:ss.fff}] [Процесс
{processId}] {message}");
                });
                // Каждая часть проверяется в отдельной задаче
                tasks.Add(lambdaDelegate(parts[i], symbol, progress,
cancellationSource.Token));
            }

            bool found = false;
            // Ждём, когда любая задача найдёт символ
            while (tasks.Count > 0)
            {
                var completedTask = await Task.WhenAny(tasks);

                if ((bool)await completedTask)
                {
                    cancellationSource.Cancel(); // Остальные задачи прервутся
                    found = true;
                    break;
                }
            }


            const string HAPPYMESSAGE = "GG!";
            const string SADMESSAGE = "Not GG...";

            string endmessage = found ? HAPPYMESSAGE : SADMESSAGE;

            Console.WriteLine(endmessage);
        }
    }
}
```

2.2. Описание кода

1. Пользовательский асинхронный делегат

public delegate Task<bool?> AsyncDelegate(string str, char symbol, IProgress<string> progress, CancellationToken cancellationToken);

- Принимает строку, символ, интерфейс прогресса и токен отмены
- Возвращает Task<bool?> для асинхронной работы

2. Ключевые механизмы

Разделение строки:

- Метод SplitString() делит исходную строку на 4 части для параллельной обработки

Лямбда-делегат:

- Поиск символа в каждой части с посимвольной проверкой
- Отчет о прогрессе через IProgress<string>
- Поддержка отмены через CancellationToken
- Искусственная задержка Task.Delay() для эмуляции длительной операции

Управление задачами:

- Создание отдельной задачи для каждой части строки
- Использование Task.WhenAny() для ожидания первой завершенной задачи
- Отмена остальных задач при успешном нахождении символа

3. Результат работы

- **"GG!"** - если символ найден в любой из частей
- **"Not GG..."** - если символ не найден ни в одной части

2.3. Результат работы программы

[17:38:36.407] [Процесс 1] Шаг 1 из 44 возможных

[17:38:36.398] [Процесс 1] Start

[17:38:36.423] [Процесс 2] Start

[17:38:36.430] [Процесс 2] Шаг 1 из 44 возможных

[17:38:36.436] [Процесс 3] Start

[17:38:36.465] [Процесс 3] Шаг 1 из 44 возможных

[17:38:36.471] [Процесс 4] Start

[17:38:36.476] [Процесс 4] Шаг 1 из 2 возможных

[17:38:36.948] [Процесс 1] Шаг 2 из 44 возможных

[17:38:36.948] [Процесс 3] Шаг 2 из 44 возможных

[17:38:36.948] [Процесс 2] Шаг 2 из 44 возможных

[17:38:36.948] [Процесс 4] Шаг 2 из 2 возможных

[17:38:37.452] [Процесс 2] Шаг 3 из 44 возможных

[17:38:37.452] [Процесс 4] Finish

[17:38:37.452] [Процесс 1] Шаг 3 из 44 возможных

[17:38:37.452] [Процесс 3] Шаг 3 из 44 возможных

[17:38:37.956] [Процесс 1] Шаг 4 из 44 возможных

[17:38:37.956] [Процесс 2] Шаг 4 из 44 возможных

[17:38:37.956] [Процесс 3] Шаг 4 из 44 возможных

[17:38:38.460] [Процесс 3] Шаг 5 из 44 возможных

[17:38:38.460] [Процесс 1] Шаг 5 из 44 возможных

[17:38:38.460] [Процесс 2] Шаг 5 из 44 возможных

[17:38:38.963] [Процесс 3] Шаг 6 из 44 возможных

[17:38:38.963] [Процесс 1] Шаг 6 из 44 возможных

[17:38:38.963] [Процесс 2] Шаг 6 из 44 возможных

[17:38:39.464] [Процесс 2] Шаг 7 из 44 возможных

[17:38:39.464] [Процесс 3] Шаг 7 из 44 возможных

[17:38:39.464] [Процесс 1] Шаг 7 из 44 возможных

[17:38:39.966] [Процесс 1] Шаг 8 из 44 возможных

[17:38:39.966] [Процесс 3] Шаг 8 из 44 возможных

[17:38:39.966] [Процесс 2] Шаг 8 из 44 возможных

[17:38:40.471] [Процесс 2] Шаг 9 из 44 возможных

[17:38:40.471] [Процесс 1] Шаг 9 из 44 возможных

[17:38:40.471] [Процесс 3] Шаг 9 из 44 возможных

[17:38:40.971] [Процесс 3] Шаг 10 из 44 возможных

[17:38:40.971] [Процесс 2] Шаг 10 из 44 возможных

[17:38:40.971] [Процесс 1] Шаг 10 из 44 возможных

[17:38:41.475] [Процесс 3] Шаг 11 из 44 возможных

[17:38:41.475] [Процесс 1] Шаг 11 из 44 возможных

[17:38:41.475] [Процесс 2] Шаг 11 из 44 возможных

[17:38:41.979] [Процесс 1] Шаг 12 из 44 возможных

[17:38:41.979] [Процесс 2] Шаг 12 из 44 возможных

[17:38:41.979] [Процесс 3] Шаг 12 из 44 возможных

[17:38:42.483] [Процесс 3] Шаг 13 из 44 возможных

[17:38:42.483] [Процесс 1] Шаг 13 из 44 возможных

[17:38:42.483] [Процесс 2] Шаг 13 из 44 возможных

[17:38:42.985] [Процесс 2] Шаг 14 из 44 возможных

[17:38:42.985] [Процесс 1] Шаг 14 из 44 возможных

[17:38:42.985] [Процесс 3] Шаг 14 из 44 возможных

[17:38:43.491] [Процесс 3] Шаг 15 из 44 возможных

[17:38:43.491] [Процесс 1] Шаг 15 из 44 возможных

[17:38:43.491] [Процесс 2] Шаг 15 из 44 возможных

[17:38:43.992] [Процесс 2] Шаг 16 из 44 возможных

[17:38:43.992] [Процесс 1] Шаг 16 из 44 возможных

[17:38:43.992] [Процесс 3] Шаг 16 из 44 возможных

[17:38:44.499] [Процесс 3] Шаг 17 из 44 возможных

[17:38:44.499] [Процесс 1] Шаг 17 из 44 возможных

[17:38:44.499] [Процесс 2] Шаг 17 из 44 возможных

[17:38:45.003] [Процесс 3] Шаг 18 из 44 возможных

[17:38:45.003] [Процесс 1] Шаг 18 из 44 возможных

[17:38:45.003] [Процесс 2] Шаг 18 из 44 возможных

[17:38:45.505] [Процесс 3] Шаг 19 из 44 возможных

[17:38:45.505] [Процесс 2] Шаг 19 из 44 возможных

[17:38:45.505] [Процесс 1] Шаг 19 из 44 возможных

[17:38:46.010] [Процесс 2] Шаг 20 из 44 возможных

[17:38:46.010] [Процесс 3] Шаг 20 из 44 возможных

[17:38:46.010] [Процесс 1] Шаг 20 из 44 возможных

[17:38:46.513] [Процесс 2] Шаг 21 из 44 возможных

[17:38:46.513] [Процесс 1] Шаг 21 из 44 возможных

[17:38:46.513] [Процесс 3] Шаг 21 из 44 возможных

[17:38:47.018] [Процесс 3] Шаг 22 из 44 возможных

[17:38:47.018] [Процесс 1] Шаг 22 из 44 возможных

[17:38:47.018] [Процесс 2] Шаг 22 из 44 возможных

[17:38:47.520] [Процесс 1] Шаг 23 из 44 возможных

[17:38:47.520] [Процесс 2] Шаг 23 из 44 возможных

[17:38:47.520] [Процесс 3] Шаг 23 из 44 возможных

[17:38:48.023] [Процесс 2] Шаг 24 из 44 возможных

[17:38:48.023] [Процесс 3] Шаг 24 из 44 возможных

[17:38:48.023] [Процесс 1] Шаг 24 из 44 возможных

[17:38:48.526] [Процесс 1] Шаг 25 из 44 возможных

[17:38:48.526] [Процесс 3] Шаг 25 из 44 возможных

[17:38:48.526] [Процесс 2] Шаг 25 из 44 возможных

[17:38:49.031] [Процесс 2] Шаг 26 из 44 возможных

[17:38:49.031] [Процесс 3] Шаг 26 из 44 возможных

[17:38:49.031] [Процесс 1] Шаг 26 из 44 возможных

[17:38:49.531] [Процесс 1] Шаг 27 из 44 возможных

[17:38:49.531] [Процесс 2] Шаг 27 из 44 возможных

[17:38:49.531] [Процесс 3] Шаг 27 из 44 возможных

[17:38:50.036] [Процесс 2] Шаг 28 из 44 возможных

[17:38:50.036] [Процесс 3] Шаг 28 из 44 возможных

[17:38:50.036] [Процесс 1] Шаг 28 из 44 возможных

[17:38:50.539] [Процесс 2] Шаг 29 из 44 возможных

[17:38:50.539] [Процесс 3] Шаг 29 из 44 возможных

[17:38:50.539] [Процесс 1] Шаг 29 из 44 возможных

[17:38:51.044] [Процесс 3] Шаг 30 из 44 возможных

[17:38:51.045] [Процесс 2] Шаг 30 из 44 возможных

[17:38:51.045] [Процесс 1] Шаг 30 из 44 возможных

[17:38:51.549] [Процесс 1] Шаг 31 из 44 возможных

[17:38:51.549] [Процесс 3] Шаг 31 из 44 возможных

[17:38:51.549] [Процесс 2] Шаг 31 из 44 возможных

[17:38:52.055] [Процесс 3] Шаг 32 из 44 возможных

[17:38:52.055] [Процесс 2] Шаг 32 из 44 возможных

[17:38:52.055] [Процесс 1] Шаг 32 из 44 возможных

[17:38:52.561] [Процесс 2] Шаг 33 из 44 возможных

[17:38:52.561] [Процесс 1] Шаг 33 из 44 возможных

[17:38:52.561] [Процесс 1] Найдено!

[17:38:52.561] [Процесс 3] Шаг 33 из 44 возможных

GG!

3. Контрольные вопросы

1. Поясните назначение типа IAsyncResult.
    a. Интерфейс, который представляет состояние асинхронной операции. Он используется в модели асинхронного программирования BeginInvoke/EndInvoke для управления асинхронными вызовами методов через делегаты.
2. Для чего используется метод Thread.Sleep( )?
    a. Это нужно для:
        i. Имитации задержек.
        ii. Снижения нагрузки на процессор в циклах опроса.
        iii. Синхронизации потоков (хотя это ненадежный метод, лучше переходить на современные рельсы с async и await).
3. Поясните механизм возврата значения из метода асинхронного делегата.
    a. Для получения результата из асинхронно вызванного метода через делегат используется метод EndInvoke, который:
        i. Принимает объект IAsyncResult, возвращенный из BeginInvoke.
        ii. Блокирует поток до завершения операции, если она еще не завершена.
        iii. Возвращает результат работы метода.
4. Как произвести возврат более одного значения из метода?
    a. Кортежи
    b. Out параметры
    c. Модификация переданных переменных
    d. Возврат объекта класса/структуры с несколькими полями.
5. Какая разница существует между библиотечными делегатами, пользовательскими типами делегатов и лямбда-выражениями?

Являются ли эти делегаты взаимозаменяемыми при реализации асинхронного вызова методов?

a. Разница между библиотечными делегатами, пользовательскими типами делегатов и лямбда-выражениями заключается в их объявлении.
b. Все эти делегаты могут использоваться для асинхронных вызовов через BeginInvoke/EndInvoke, если их сигнатуры совместимы, но лямбда-выражения, захватывающие внешние переменные (замыкания), требуют осторожности из-за рисков синхронизации в многопоточной среде.