Ильяхова Алиса, Б9122-02.03.01сцт

Лабораторная работа №5. Применение класса Thread

Вариант 9

1. Цель и задачи работы

Цель:

Научиться использовать класс Thread для организации многопоточного приложения.

Задачи:

- Научиться создавать и запускать потоки с использованием класса Thread;
- Научиться передавать параметры в потоки;
- Научиться организовывать ожидание завершения потоков.

2. Реализация индивидуального задания

2.1. Условие варианта 9

Согласно таблице индивидуальных заданий (стр. 19):

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

2.2. Объявление метода шифрования

Реализован метод EncryptString, выполняющий побайтовое шифрование:

```
static void EncryptString(string input, int shift)
    {
        if (input == null) return;

        Console.WriteLine($"[Поток {Thread.CurrentThread.ManagedThreadId}] Начало
шифрования строки: \"{input}\"");
        Thread.Sleep(2000); // Имитация долгой операции

        char[] buffer = new char[input.Length];
        for (int i = 0; i < input.Length; i++)
```

```
        {
            buffer[i] = (char)(input[i] + shift);
        }

        string encrypted = new string(buffer);
        Console.WriteLine($"[Поток {Thread.CurrentThread.ManagedThreadId}] Результат:
\"{encrypted}\"");
        Console.WriteLine(new string('-', 50));
    }
```

- Метод **не возвращает значение** (void), так как результат выводится напрямую в консоль (в соответствии с требованиями ЛР №5).
- Добавлена имитация долгой операции через Thread.Sleep(2000).

2.3. Создание и запуск потоков

- Создан массив тестовых данных: пары (строка, сдвиг).
- Для каждой пары создан отдельный объект Thread.
- Использовано **лямбда-выражение** для передачи параметров:threads[i] = new Thread(() => EncryptString(testData[index].text, testData[index].shift));

- Переменная index захвачена для избежания ошибки захвата циклической переменной.

2.4. Ожидание завершения

- После запуска всех потоков выполнено ожидание их завершения через Join():foreach (var thread in threads)
  thread.Join();

- Это гарантирует, что основной поток не завершится раньше фоновых.

3. Ответы на контрольные вопросы

1. **В каком пространстве имен определен класс Thread? Поясните назначение основных методов и свойств класса.**

Класс Thread определён в пространстве имён System.Threading.

Основные методы:

a. Start() — запускает поток;
b. Join() — блокирует вызывающий поток до завершения текущего;

c. Sleep() — приостанавливает выполнение потока. Основные свойства:
d. ManagedThreadId — уникальный идентификатор управляемого потока;
e. IsAlive — указывает, выполняется ли поток.

2. **Какие существуют способы передачи параметров в поток?**
   a. Через **замыкание** (лямбда-выражение или анонимный метод);
   b. Через **параметризованный конструктор** Thread(ParameterizedThreadStart) и метод с параметром object;
   c. Через **члены класса** (если поток запускает метод экземпляра).
3. **Опишите механизм синхронизации завершения потоков.**

Метод Join() позволяет дождаться завершения потока. Без него основной поток может завершиться раньше, и результаты фоновых потоков не будут видны.

4. **Как получить идентификатор текущего потока?**

Через свойство Thread.CurrentThread.ManagedThreadId.

4. Экранные формы и листинг программы

4.1. Консольный вывод программы

```
=== Лабораторная работа №5. Вариант 9 ===
Применение класса Thread

[Поток 11] Начало шифрования строки: "Secret"
[Поток 10] Начало шифрования строки: "World"
[Поток 9] Начало шифрования строки: "Hello"
[Поток 12] Начало шифрования строки: "Message"
[Поток 12] Результат: "Oguucig"
[Поток 11] Результат: "Zljyl{"
[Поток 10] Результат: "\twqi"
[Поток 9] Результат: "Khoor"
4 ------------------------------------------------

 Все потоки завершены.
```

Порядок вывода может отличаться, так как потоки выполняются параллельно.

4.2. Полный листинг программы с комментариями

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

```
// Лабораторная работа №5. Применение класса Thread
// Вариант 9: шифрование строки с заданным сдвигом

class Program
{
    // Метод шифрования строки
    static void EncryptString(string input, int shift)
    {
        if (input == null) return;

        Console.WriteLine($"[Поток {Thread.CurrentThread.ManagedThreadId}] Начало
шифрования строки: \"{input}\"");
        Thread.Sleep(2000); // Имитация долгой операции

        char[] buffer = new char[input.Length];
        for (int i = 0; i < input.Length; i++)
        {
            buffer[i] = (char)(input[i] + shift);
        }

        string encrypted = new string(buffer);
        Console.WriteLine($"[Поток {Thread.CurrentThread.ManagedThreadId}] Результат:
\"{encrypted}\"");
        Console.WriteLine(new string('-', 50));
    }

    static void Main()
    {
        Console.WriteLine("=== Лабораторная работа №5. Вариант 9 ===");
        Console.WriteLine("Применение класса Thread\n");

        // Тестовые данные: строки и сдвиги
        var testData = new (string text, int shift)[]
        {
            ("Hello", 3),
            ("World", 5),
            ("Secret", 7),
            ("Message", 2)
        };

        // Создаём массив потоков
        Thread[] threads = new Thread[testData.Length];

        // Запускаем потоки
        for (int i = 0; i < testData.Length; i++)
        {
            int index = i; // захват переменной для корректной передачи в замыкание
            threads[i] = new Thread(() =>
            {
                EncryptString(testData[index].text, testData[index].shift);
```

```
            });
            threads[i].Start();
        }

        // Ожидаем завершения всех потоков
        foreach (var thread in threads)
        {
            thread.Join();
        }

        Console.WriteLine("\n Все потоки завершены.");
    }
}
```

5. Вывод

В ходе выполнения лабораторной работы №5 были:

- Реализован метод шифрования строки согласно варианту 9;
- Созданы и запущены несколько потоков с использованием класса Thread;
- Продемонстрирована передача параметров через лямбда-выражение;
- Организовано ожидание завершения всех потоков через Join();
- Показан параллельный вывод информации с указанием идентификаторов потоков.