**ЛАБОРАТОРНА РОБОТА № 1**

***Взаємодія між процесами. Розподіл даних між процесами. Робота з файлами, які відображаються у пам’ять.***

**Хід роботи:**

Посилання на репозиторій:

https://github.com/andreylion06/SystemNetworkProgramming-Lab1

***Завдання 1***:

Необхідно написати дві програми (**три**), які будуть мати спільні дані  та одночасно до них звертатися.

Існує кілька механізмів реалізації спільного доступу до даних різних процесів.

Скористаємося одним з них, найбільш зручним - проектуванням файлу в пам'ять.

Одна програма буде сортувати дані у файлі, а інша відображати вміст цього файлу. Працювати обидва процеси будуть одночасно. **Третя програма** буде створювати (або заповнювати по новому) масив випадкових чисел.

Створіть файл data.dat. У ньому мають бути записані числа, згенеровані випадковим чином. Кількість чисел - 20-30 штук. Діапазон значень: від 10 до 100. (Це саме числа, а не символьні рядки зберігають ASCII коди цифр !!!)

**Програма №1. "Сортування даних" (консольна)**

Беремо за основу програму "Hello windows". Включаємо обробку події натискання клавіші, і відстежуємо в ньому натискання пробілу. Якщо користувач натиснув пробіл, значить починаємо сортування даних.  
Виконуємо проектування файлу в пам'ять. Використовуємо для цього створений файл data.dat. В результаті отримаємо доступ до даних як до звичайного одновимірного масиву.

Виконуємо сортування масиву, будь-яким з методів сортування. Вставте 1-но секундну затримку для кожної ітерації сортування масиву, це дозволить потім наочніше побачити процес сортування.  
По закінченню сортування, програма виводить у вікно, рядок «Робота завершена».

**Програма №2. «Виведення файлу даних у вікно» (віконна)**

Виконуємо проектування файлу в пам'ять. Використовуємо для цього створений файл data.dat. В результаті отримаємо доступ до даних як до звичайного одновимірного масиву. Цей же файл проектує в пам'ять попередня програма.  
Створюємо таймер на 0.5 секунди. При отриманні повідомлення від таймера, виконуємо висновок всього масиву в вікно. Передбачте коректний перевивід даних у вікно, без накладень. У вікно виводиться не числа з масиву, а рядки одного і того ж символу, наприклад «\*», в кількості, що дорівнює числу з масиву.

Запускаємо на виконання обидві програми одночасно. Коли друга програма запустилася і виконує висновок даних у вікно (виводить поки одну й ту ж саму картинку кожні пів секунди), натискаємо пробіл в першій програмі і вона починає сортувати масив. При цьому, так як вони дані беруть з одного і того ж файлу (обидві проектували його собі на згадку), то перша вносить зміни переставляючи дані при сортуванні, а друга виводить з себе у вікно і ми бачимо хід процесу сортування. Тимчасову затримку в першій програмі можна при потребі збільшити.

Ці дві програми демонструють можливість організації спільного доступу процесів до одних і тих самих даних. Так само демонструється механізм проектування файлу в пам'ять, як один з найкращих методів доступу до файлу.

***Звіт про програму:***

Програми в папці "ConsoleTask1" спрямовані на демонстрацію можливостей спільного доступу до даних різних процесів, а також на використання механізму проектування файлу в пам'ять для забезпечення цього доступу. Основними функціями програм є генерація випадкових чисел у файл, сортування цих чисел та виведення відсортованих даних у вікно у вигляді символьних рядків.

Рішення реалізоване за допомогою 3 консольних додатків використовуючи мову програмування C#. Данні додатки мають змогу працювати одночасно.

Структура програми:

* ConsoleTask1Generation: Ця програма відповідає за генерацію випадкових чисел та запис їх у файл "data.dat". Вона має основний цикл, який виконується, доки користувач не вибере вихід з програми. Кожне випадкове число записується у файл, після чого користувач має можливість продовжити генерацію або завершити програму.
* ConsoleTask1Output: Ця програма відповідає за виведення даних з файлу "data.dat" у вікно у вигляді символьних рядків, які представляють числа у вигляді зірочок. Вона використовує механізм проектування файлу в пам'ять для доступу до даних, а також використовує таймер для періодичного оновлення вмісту вікна.
* ConsoleTask1Sorting: Ця програма відповідає за сортування чисел з файлу "data.dat" з затримкою між кроками сортування. Вона також використовує механізм проектування файлу в пам'ять для доступу до даних. Після натискання пробілу користувачем програма розпочинає сортування чисел у файлі з використанням алгоритму сортування бульбашкою, затримуючи виконання кожного кроку сортування на 1 секунду для наочного відображення процесу.

***Лістинг програми:***

using System;

using System.IO;

// Ім'я файлу: ConsoleTask1Generation.Program.cs

// Ремарка: Виконує генерацію {numberOfRandomNumbers} чисел у файл "data.dat"

// Автор: Андрій Сахно

class Program

{

static void Main(string[] args)

{

try

{

string solutionRoot = Directory.GetParent(Directory.GetCurrentDirectory())!.Parent!.Parent!.Parent!.FullName;

string filePath = Path.Combine(solutionRoot, "data.dat");

int numberOfRandomNumbers = 25; // Number of random numbers in the file

Random random = new Random(); // Create a single instance of Random

// Loop until 'q' is pressed

while (true)

{

using (StreamWriter writer = new StreamWriter(filePath))

{

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

{

int randomNumber = random.Next(10, 101); // Random number between 10 and 100

writer.WriteLine(randomNumber);

}

}

Console.WriteLine($"Written {numberOfRandomNumbers} random numbers to the file {filePath}.");

Console.WriteLine("Press Enter for the next iteration or 'q' to exit:");

string? input = Console.ReadLine();

if (!string.IsNullOrEmpty(input) && input.ToLower() == "q")

break;

}

}

catch (Exception ex)

{

Console.WriteLine($"An error occurred: {ex.Message}");

}

}

}

// Кінець файлу

using System;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

// Ім'я файлу: ConsoleTask1Output.Program.cs

// Ремарка: Виводить числа з файлу "data.dat" у форматі зірочок

// Автор: Андрій Сахно

class Program

{

static async Task Main(string[] args)

{

try

{

string solutionRoot = Directory.GetParent(Directory.GetCurrentDirectory())!.Parent!.Parent!.Parent!.FullName;

string filePath = Path.Combine(solutionRoot, "data.dat");

// Read numbers from the file

while (true)

{

int[] numbers = File.ReadAllLines(filePath).Select(int.Parse).ToArray();

foreach (var item in numbers)

{

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

{

Console.Write("\*");

}

Console.WriteLine();

}

await Task.Delay(500);

Console.Clear();

}

}

catch (Exception ex)

{

Console.WriteLine($"An error occurred: {ex.Message}");

}

}

}

// Кінець файлу

using System;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

// Ім'я файлу: ConsoleTask1Sorting.Program.cs

// Ремарка: Виконує сортування чисел з файлу "data.dat" з затримкою

// Автор: Андрій Сахно

class Program

{

static async Task Main(string[] args)

{

try

{

string solutionRoot = Directory.GetParent(Directory.GetCurrentDirectory())!.Parent!.Parent!.Parent!.FullName;

string filePath = Path.Combine(solutionRoot, "data.dat");

// Read numbers from the file

int[] numbers = File.ReadAllLines(filePath).Select(int.Parse).ToArray();

// Loop until 'q' is pressed

while (true)

{

Console.Write("Press Space to start sorting ==> ");

ConsoleKeyInfo keyInfo = Console.ReadKey(true);

Console.WriteLine();

if (keyInfo.Key == ConsoleKey.Spacebar)

{

Console.Clear();

break;

}

}

await SortAndWriteToFileWithDelay(numbers, filePath);

Console.WriteLine($"\n\n\*\*\* Numbers in the file {filePath} have been sorted. \*\*\*");

}

catch (Exception ex)

{

Console.WriteLine($"An error occurred: {ex.Message}");

}

}

static async Task SortAndWriteToFileWithDelay(int[] array, string filePath)

{

for (int i = 0; i < array.Length - 1; i++)

{

for (int j = 0; j < array.Length - i - 1; j++)

{

if (array[j] > array[j + 1])

{

Console.WriteLine($"'{array[j]}' and '{array[j + 1]}' have been swapped");

Swap(ref array[j], ref array[j + 1]);

File.WriteAllLines(filePath, array.Select(n => n.ToString()));

await Task.Delay(1000);

}

}

}

}

static void Swap(ref int a, ref int b)

{

int temp = a;

a = b;

b = temp;

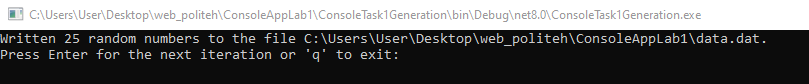
}

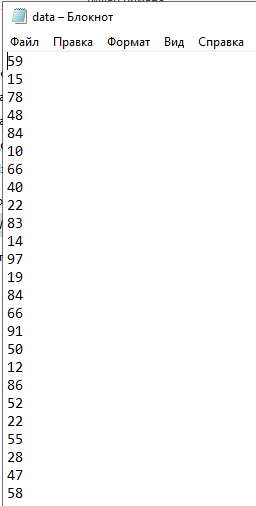
}

// Кінець файлу

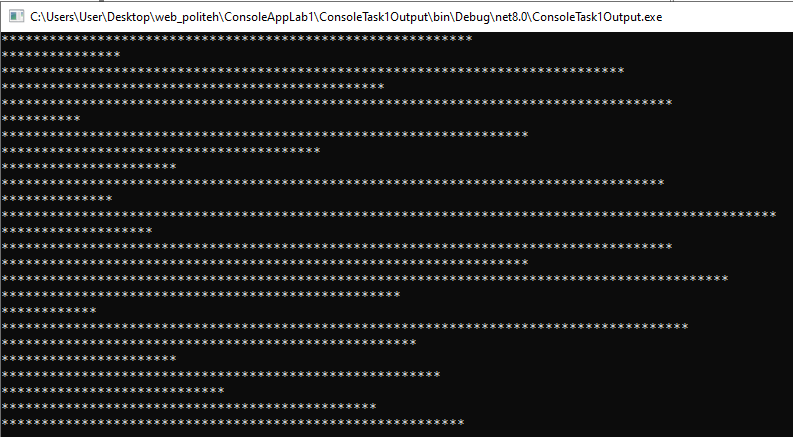
***Результат виконання:***

Генерую числа у файл data.dat за допомогою ConsoleTask1Sorting.Program.cs.

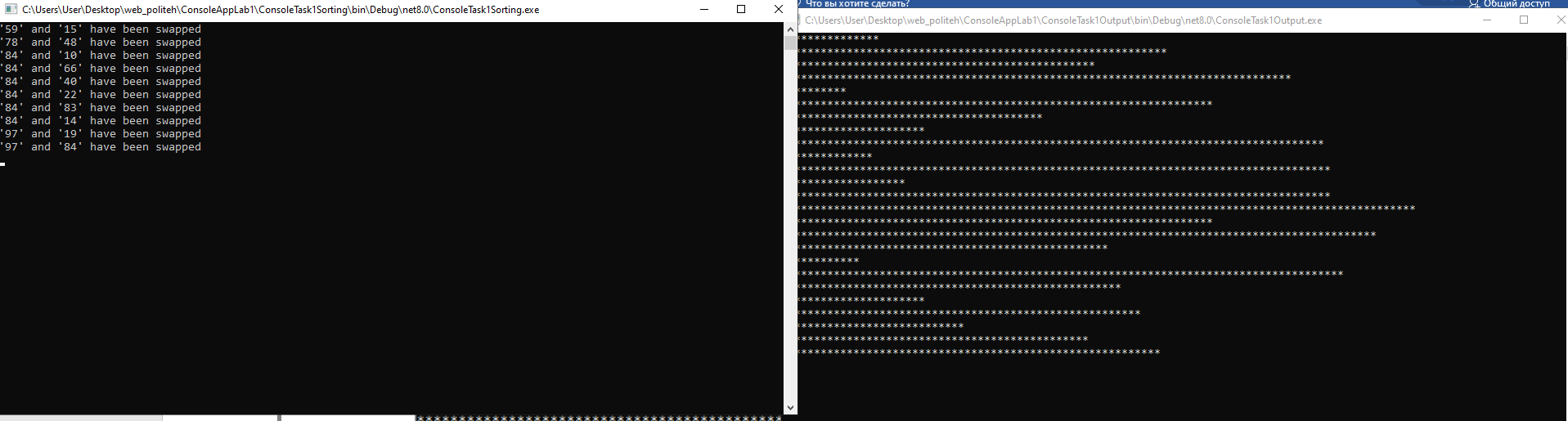
******

******

Запускаю ConsoleTask1Output.Program.cs для відображення чисел з файлу у форматі зірочок.



Також запускаю ConsoleTask1Sorting.Program.cs, при цьому всі три програми працюють одночасно та не мають конфліктів. Можу спостерігати за процесом сортування у консольі виводу в форматі зірочок.



***Завдання 2***:

Для коректної роботи зі спільними даними у цих двох програмах потрібно додати **синхронізацію потоків**, які можуть одночасно звертатися до спільних даних.

Для організації такої синхронізації потрібно використати об'єкт ядра ОС **mutex або semaphor**, або інший синхронізуючий об'єкт, а також **функції очікування** (наприклад, **WaitForSinglJbject()**).

Також обов'язковим є використання **обробки виняткових ситуацій** в роботі вище описаних трьох програм. Бо, некоректна робота будь якої з трьох, викличе неправильну роботу інших, через блокування спільних даних.

Для обробки виняткових ситуацій, необхідно правильно визначити **критичні секції коду** усіх написаних програм.

***Звіт про програму:***

Програма "ConsoleTask2" розроблена з метою синхронізації процесів сортування та виведення даних з файлу "data.dat" у вікно у вигляді символьних рядків. Для досягнення цієї мети використовується об'єкт SemaphoreSlim для забезпечення взаємного виключення доступу до спільних даних - числового масиву та файлу.

Рішення реалізоване за допомогою консольного додатку використовуючи мову програмування C#.

Структура програми:

* SemaphoreSlim: Для забезпечення синхронізації використовується об'єкт SemaphoreSlim з кількістю дозволених потоків 2. Це дозволяє виконувати операції сортування та виведення даних одночасно, але забороняє виконання цих операцій паралельно.
* Cancellation Token: Використовується CancellationTokenSource для обробки виняткових ситуацій та виходу з програми в разі виникнення помилки. Це забезпечує коректне завершення роботи програми при будь-яких виняткових обставинах.
* Main метод: Основний метод програми, який викликається при запуску. Виконує ініціалізацію шляху до файлу "data.dat", створює завдання для виконання операцій сортування та виведення даних, та очікує завершення обох завдань перед завершенням роботи програми.
* Output метод: Метод відповідає за виведення вмісту файлу "data.dat" у вікно у вигляді символьних рядків. Використовує SemaphoreSlim для забезпечення взаємного виключення доступу до файлу. Обробляє виняткові ситуації та виходить з програми в разі помилки.
* Sort метод: Метод відповідає за сортування чисел у файлі "data.dat". Використовує SemaphoreSlim для забезпечення взаємного виключення доступу до файлу. Обробляє виняткові ситуації та виходить з програми в разі помилки.
* SortAndWriteToFileWithDelay метод: Метод виконує сортування чисел за допомогою алгоритму бульбашкового сортування та записує відсортований масив у файл. Використовує CancellationToken для перевірки стану скасування операцій у разі виняткової ситуації.

***Лістинг програми:***

using System;

using System.IO;

using System.Threading;

// Ім'я файлу: ConsoleTask2.Program.cs

// Ремарка: Сортування та вивід чисел у форматі зірочок синхронізовані за допомогою SemaphoreSlim.

// Реалізована обробка виняткових ситуацій, у випадку якщо один з потоків стикнеться з

// винятковою ситуацією - зупиняться обидва

// Автор: Андрій Сахно

class Program

{

static SemaphoreSlim semaphore = new SemaphoreSlim(2);

static CancellationTokenSource cancellationTokenSource = new CancellationTokenSource();

static string FilePath { get; set; } = String.Empty;

static async Task Main(string[] args)

{

string solutionRoot = Directory.GetParent(Directory.GetCurrentDirectory())!.Parent!.Parent!.Parent!.FullName;

FilePath = Path.Combine(solutionRoot, "data.dat");

Task output = Output();

Task sort = Sort();

await Task.WhenAll(output, sort);

}

static public async Task Output()

{

try

{

await semaphore.WaitAsync(cancellationTokenSource.Token);

do

{

Console.Clear();

int[] numbers = File.ReadAllLines(FilePath).Select(int.Parse).ToArray();

foreach (var item in numbers)

{

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

{

Console.Write("\*");

}

Console.WriteLine();

}

await Task.Delay(500);

} while (!cancellationTokenSource.Token.IsCancellationRequested);

}

catch (Exception ex)

{

cancellationTokenSource.Cancel();

Console.WriteLine($"Exception in Output: {ex.Message}");

}

}

static public async Task Sort()

{

try

{

await semaphore.WaitAsync(cancellationTokenSource.Token);

int[] numbers = File.ReadAllLines(FilePath).Select(int.Parse).ToArray();

await SortAndWriteToFileWithDelay(numbers);

semaphore.Release();

if(!cancellationTokenSource.Token.IsCancellationRequested)

Console.WriteLine($"\n\n~~~ Numbers in the file {FilePath} have been sorted. ~~~");

cancellationTokenSource.Cancel();

Console.WriteLine($"Stoping the semaphore...\n\n");

}

catch (Exception ex)

{

cancellationTokenSource.Cancel();

Console.WriteLine($"Exception in Sort: {ex.Message}");

}

}

static private async Task SortAndWriteToFileWithDelay(int[] array)

{

for (int i = 0; i < array.Length - 1; i++)

{

for (int j = 0; j < array.Length - i - 1; j++)

{

if (cancellationTokenSource.Token.IsCancellationRequested)

return;

if (array[j] > array[j + 1])

{

Swap(ref array[j], ref array[j + 1]);

File.WriteAllLines(FilePath, array.Select(n => n.ToString()));

await Task.Delay(1000);

}

}

if (cancellationTokenSource.Token.IsCancellationRequested)

return;

}

}

static private void Swap(ref int a, ref int b)

{

int temp = a;

a = b;

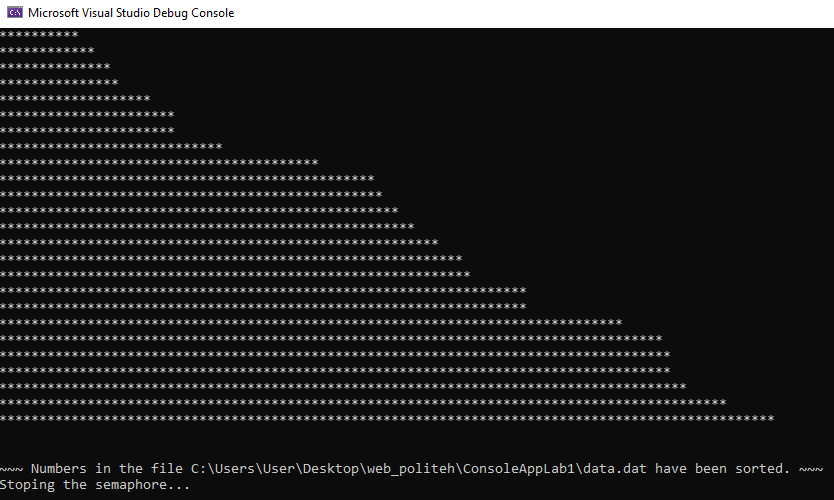
b = temp;

}

}

// Кінець файлу

***Результат виконання:***



Сортування та вивід здійснюються синхронно за допомогою SemaphoreSlim в одному консольному проекті.

***Висновок:*** У результаті виконання лабораторної роботи було розроблено програму "ConsoleAppLab1", яка успішно демонструє використання механізмів синхронізації потоків та обробки виняткових ситуацій у мові програмування C#. Використання SemaphoreSlim та CancellationToken дозволяє ефективно управляти спільним доступом до даних та забезпечує надійність роботи програм.