**ЛАБОРАТОРНА РОБОТА № 5**

***Динамічно зв’язувані біліотеки***

**Хід роботи:**

Посилання на репозиторій:

<https://github.com/andreylion06/SystemNetworkProgramming-Lab5>

***Завдання***:

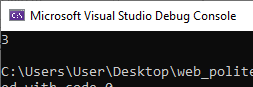
1. Написати програмний продукт, який буде мати об'єктно-орієнтовану архітектуру. Обов'язкове створення та реалізація двох класів. Перший - інтерфейс, другий - обчислення.

2. Кожен клас має розташовуватися в окремій DLL.

3. DLL яка реалізує інтерфейс має завантажуватися разом з основною програмою.

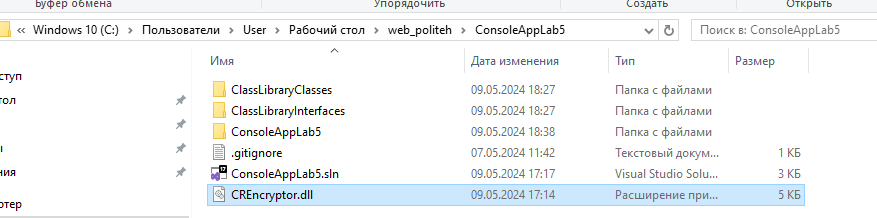
4. DLL яка реалізує обчислення має завантажуватися пізніше, вже під час роботи програмного продукту, за вимогою користувача. А також після використання відвантажуватися з пам'яті. Перед завантаженням необхідно перевірити чи є вже бібліотека у пам'яті.

Успішне використання методу з MyCalculator:

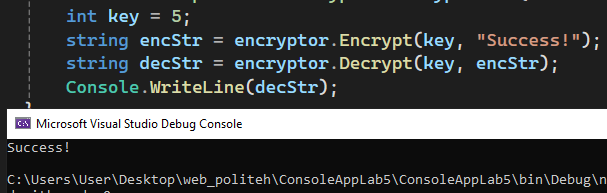


5. Підключити та використати будь які функції для прикладу, будь яку не власну бібліотеку.

Підвантажив зовнішню бібліотеку CREncryptor в корінь рішення:

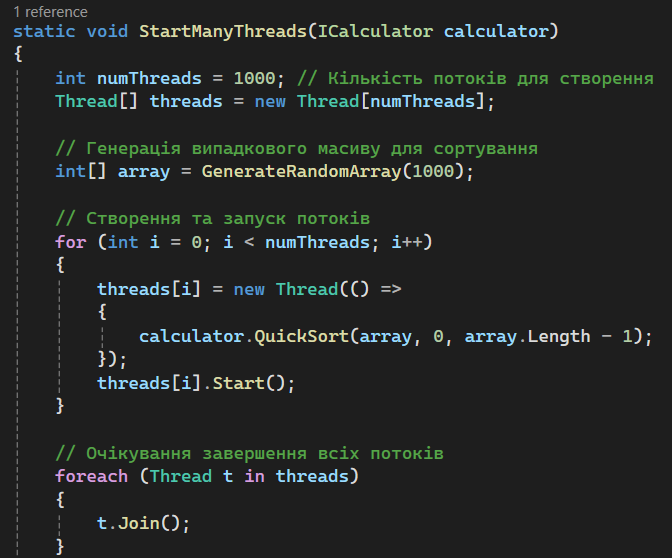


Успішне шифрування та дешифрування з допомогою CREncryptor:

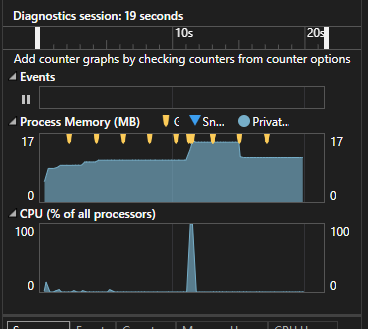


6. При реалізації класу обчислень, використати алгоритм повного перебору, або сортування qsort, та використати породження багатьох потоків. Вивчити поведінку системи при граничному використанні пам'яті та породжених потоків. Показати графіки використання ресурсів. Проаналізувати залежності в них.

У StartManyThreads(ICalculator calculator) запускаю 1000 потоків, де сортую масив із 100 елементів.



Після виконання цього метода маю наступний графік використання ресурсів:



Бачу, що з момента запуску потоків відсоток використання CPU досяг максимуму – 100%, використання оперативної пам’яті зібльшилось. Після закінчення виконанння функціоналу в усіх потоках, використання ресурсів нормалізувалось та стало стабільним як до їх запуску.

***Звіт про програму:***

Програма "ConsoleAppLab5" створена з метою демонстрації реалізації об'єктно-орієнтованої архітектури та використання динамічного завантаження бібліотек у середовищі .NET. Ця програма дозволяє завантажувати та взаємодіяти з DLL-файлами упродовж її роботи.

Структура програми:

* MyCalculator.cs: Клас MyCalculator визначає реалізацію інтерфейсу ICalculator, який містить методи для обчислення суми, факторіалу, чисел Фібоначчі та сортування масиву.
* AssemblyLoader.cs: Клас AssemblyLoader надає методи для завантаження, створення екземплярів та вивантаження збірок (assembly) з власного проекту або за абсолютним шляхом. Він дозволяє взаємодіяти з класами з завантажених збірок.
* Program.cs: Головний клас програми містить методи для демонстрації функціоналу AssemblyLoader. Тут реалізовано три різні сценарії: завантаження та використання класу MyCalculator, завантаження та використання зовнішньої бібліотеки CREncryptor для шифрування тексту, а також демонстрація породження багатьох потоків для сортування масиву за допомогою класу MyCalculator.

***Лістинг програми:***

using ClassLibraryInterfaces;

using ConsoleAppLab5.AssemblyLoader;

// Ім'я файлу: Program.cs

// Ремарка: Клас Program призначений для тестування функціоналу класа AssemblyLoader

// Автор: Андрій Сахно

class Program

{

static void Main(string[] args)

{

//LoadAndTryMyAssembly();

//LoadAndTryCREncryptor();

LoadAndTryManyThreads();

}

static void LoadAndTryMyAssembly()

{

try

{

var libraryLoader = new AssemblyLoader();

libraryLoader.LoadAssemblyFromProjectInSameSolution("ClassLibraryClasses");

object? calculatorInstance = libraryLoader.GetClassInstanceFromAssembly("MyCalculator");

if (calculatorInstance != null && calculatorInstance is ICalculator)

{

ICalculator calculator = (ICalculator)calculatorInstance;

var result = calculator.Calculate(1, 2);

Console.WriteLine(result);

}

libraryLoader.UnloadAssembly();

}

catch (Exception ex)

{

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

}

}

static void LoadAndTryCREncryptor()

{

try

{

var libraryLoader = new AssemblyLoader();

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

string filePath = Path.Combine(solutionRoot, "CREncryptor.dll");

libraryLoader.LoadAssemblyByAbsolutePath("CREncryptor", filePath);

object? encryptorInstance = libraryLoader.GetClassInstanceFromAssembly("CharacterReplacementEncryptor");

if (encryptorInstance != null)

{

ICharacterReplacementEncryptor encryptor = (ICharacterReplacementEncryptor)encryptorInstance;

int key = 5;

string encStr = encryptor.Encrypt(key, "Success!");

string decStr = encryptor.Decrypt(key, encStr);

Console.WriteLine(decStr);

}

libraryLoader.UnloadAssembly();

}

catch (Exception ex)

{

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

}

}

static void LoadAndTryManyThreads()

{

try

{

var libraryLoader = new AssemblyLoader();

libraryLoader.LoadAssemblyFromProjectInSameSolution("ClassLibraryClasses");

object? calculatorInstance = libraryLoader.GetClassInstanceFromAssembly("MyCalculator");

if (calculatorInstance != null && calculatorInstance is ICalculator)

{

ICalculator calculator = (ICalculator)calculatorInstance;

Console.ReadLine();

StartManyThreads(calculator);

}

libraryLoader.UnloadAssembly();

Console.ReadLine();

}

catch (Exception ex)

{

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

}

}

static void StartManyThreads(ICalculator calculator)

{

int numThreads = 1000; // Кількість потоків для створення

Thread[] threads = new Thread[numThreads];

// Генерація випадкового масиву для сортування

int[] array = GenerateRandomArray(1000);

// Створення та запуск потоків

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

{

threads[i] = new Thread(() =>

{

calculator.QuickSort(array, 0, array.Length - 1);

});

threads[i].Start();

}

// Очікування завершення всіх потоків

foreach (Thread t in threads)

{

t.Join();

}

Console.WriteLine("All threads have finished sorting.");

}

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(1000); // Для прикладу, максимальне значення - 1000

}

return array;

}

}

// Кінець файлу

using ClassLibraryInterfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleAppLab5

{

using System;

using System.IO;

using System.Linq;

using System.Reflection;

namespace AssemblyLoader

{

// Ім'я файлу: AssemblyLoader.cs

// Ремарка: Клас AssemblyLoader призначений для завантаження та взаємодії з збірками (assembly) у .NET проекті.

// Він надає методи для завантаження збірок з власного проекту у тій самій рішенні (solution), або за абсолютним шляхом.

// Клас також дозволяє створювати екземпляри класів з завантажених збірок та розгружати їх з поточного домену застосунку.

// Автор: Андрій Сахно

public class AssemblyLoader

{

public string? AssemblyName { get; private set; }

private Assembly? \_classesAssembly;

public bool IsAssemblyLoaded()

{

return \_classesAssembly != null;

}

public bool LoadAssemblyFromProjectInSameSolution(string assemblyName)

{

if (IsAssemblyLoaded())

{

throw new Exception("Assembly is already loaded.");

}

try

{

AssemblyName = assemblyName;

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

if (solutionRoot != null)

{

string filePath = Path.Combine(solutionRoot, $@"{AssemblyName}\bin\Debug\net8.0\{AssemblyName}.dll");

if (File.Exists(filePath))

{

\_classesAssembly = Assembly.LoadFrom(filePath);

return true;

}

else

{

throw new FileNotFoundException($"Assembly {AssemblyName} not found at '{filePath}'.");

}

}

else

{

throw new DirectoryNotFoundException("Failed to find solution root directory path.");

}

}

catch (Exception ex)

{

throw new Exception($"Error loading assembly: {ex.Message}");

}

}

public bool LoadAssemblyByAbsolutePath(string assemblyName, string absolutePath)

{

if (IsAssemblyLoaded())

{

throw new Exception("Assembly is already loaded.");

}

try

{

AssemblyName = assemblyName;

if (File.Exists(absolutePath))

{

\_classesAssembly = Assembly.LoadFrom(absolutePath);

return true;

}

else

{

throw new FileNotFoundException($"Assembly not found at '{absolutePath}'.");

}

}

catch (Exception ex)

{

throw new Exception($"Error loading assembly from absolute path: {ex.Message}");

}

}

public object? GetClassInstanceFromAssembly(string className)

{

if (!IsAssemblyLoaded())

{

throw new Exception("Assembly is not loaded.");

}

try

{

return \_classesAssembly?.CreateInstance($"{AssemblyName}.{className}");

}

catch (Exception ex)

{

throw new Exception($"Error creating instance of {className}: {ex.Message}");

}

}

public void UnloadAssembly()

{

try

{

AppDomain.CurrentDomain.GetAssemblies()

.Where(a => a.FullName!.Contains(AssemblyName))

.ToList()

.ForEach(a => AppDomain.CurrentDomain.Load(a.GetName()));

}

catch (Exception ex)

{

throw new Exception($"Error unloading assembly: {ex.Message}");

}

}

}

}

}

// Кінець файлу

using ClassLibraryInterfaces;

using System.Text.Json;

namespace ClassLibraryClasses

{

// Ім'я файлу: MyCalculator.cs

// Ремарка: MyCalculator - включає в себе деякі математичні операції

// Автор: Андрій Сахно

public class MyCalculator : ICalculator

{

public double Calculate(double x, double y)

{

return x + y;

}

public int Factorial(int n)

{

if (n == 0)

return 1;

else

return n \* Factorial(n - 1);

}

public int Fibonacci(int n)

{

if (n <= 1)

return n;

else

return Fibonacci(n - 1) + Fibonacci(n - 2);

}

public void QuickSort(int[] array, int left, int right)

{

if (left < right)

{

int pivotIndex = Partition(array, left, right);

QuickSort(array, left, pivotIndex - 1);

QuickSort(array, pivotIndex + 1, right);

}

}

private int Partition(int[] array, int left, int right)

{

int pivot = array[right];

int i = left - 1;

for (int j = left; j < right; j++)

{

if (array[j] < pivot)

{

i++;

Swap(array, i, j);

}

}

Swap(array, i + 1, right);

return i + 1;

}

private void Swap(int[] array, int i, int j)

{

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

}

// Кінець файлу

namespace ClassLibraryInterfaces

{

public interface ICalculator

{

double Calculate(double x, double y);

int Factorial(int n);

int Fibonacci(int n);

void QuickSort(int[] array, int left, int right);

}

}

// Кінець файлу

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibraryInterfaces

{

public interface ICharacterReplacementEncryptor

{

string Encrypt(int key, string plainText);

string Decrypt(int key, string encryptedText);

}

}

// Кінець файлу

***Висновок:*** У результаті виконання лабораторної роботи було створено програму "ConsoleAppLab5", що демонструє можливості динамічного завантаження бібліотек та їх використання в середовищі .NET. Використовуючи клас AssemblyLoader, програма може завантажувати, створювати екземпляри та вивантажувати збірки з власного проекту або за абсолютним шляхом, що надає гнучкість та можливість розширення функціоналу без перекомпіляції основного коду. Такий підхід спрощує управління програмними компонентами та полегшує розробку програм з розширеним функціоналом.