Лабораторна робота № 5

Динамічно зв'язувані бібліотеки.

Мета: Вивчення роботи з DLL.

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.

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

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.

Виконання роботи:

Для

Інтерфейс DLL: Містить інтерфейс, який визначає методи для обчислень.

Обчислення DLL: Реалізує інтерфейс та містить методи для обчислень, а також реалізує алгоритм сортування з використанням потоків.

Консольний додаток: Завантажує інтерфейсну DLL при старті, а обчислювальну DLL за вимогою користувача. Використовує функції з обчислювальної DLL та вивантажує її після використання.

					ЖИТОМИРСЬКА ПОЛІТЕХНІКА.24.123.15.000 – Лр.5		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Тарнопольський			Звіт з лабораторної роботи №5	Літ.	Арк. 1 / Аркушів 9
Перевір.		Власенко О.В					
Реценз.						ФІКТ, гр. КІ-21-1	
Н. Контр.							
Зав.каф.		Єфіменко А.А.					

Для дослідження динамічно звязаних бібліотек нам по суті все одно, яку програму писати, тому напишено звичайних простий калькулятор:

InterfaceLibrary: Містить інтерфейс ICalculator.

```
namespace InterfaceLibrary
{
    public interface ICalculator
    {
        double Add(double a, double b);
        double Subtract(double a, double b);
        double Multiply(double a, double b);
        double Divide(double a, double b);
    }
}
```

CalculationLibrary: Містить клас Calculator, який реалізує інтерфейс ICalculator і включає алгоритм сортування з використанням потоків.

```
// Calculator.cs
using InterfaceLibrary;
using System;
using System.Threading.Tasks;

namespace CalculationLibrary
{
    public class Calculator : ICalculator
    {
        public double Add(double a, double b) => a + b;
        public double Subtract(double a, double b) => a - b;
        public double Multiply(double a, double b) => a * b;
        public double Divide(double a, double b)
        {
            if (b == 0)
                throw new DivideByZeroException("Division by zero is not allowed.");
            return a / b;
        }
    }
}
```

MainApp: Консольний додаток, який завантажує інтерфейсну DLL при старті. За вимогою користувача завантажує та вивантажує обчислювальну DLL.

```
using System;
using System.IO;
using System.Reflection;
using InterfaceLibrary;
using Newtonsoft.Json;
```

```
namespace MainApp
{
    class Program
    {
        private static ICalculator calculator;
        private static Assembly calcAssembly;

        static void Main(string[] args)
        {
            LoadInterfaceDLL();
            while (true)
            {
                Console.WriteLine("Enter a command (load, unload, add, sub, mul, div, exit):");
                string command = Console.ReadLine();
                switch (command)
                {
                    case "load":
                        LoadCalculationDLL();
                        break;
                    case "unload":
                        UnloadCalculationDLL();
                        break;
                    case "add":
                        PerformOperation((a, b) => calculator.Add(a, b));
                        break;
                    case "sub":
                        PerformOperation((a, b) => calculator.Subtract(a, b));
                        break;
                    case "mul":
                        PerformOperation((a, b) => calculator.Multiply(a, b));
                        break;
                    case "div":
                        PerformOperation((a, b) => calculator.Divide(a, b));
                        break;
                    case "exit":
                        return;
                }
            }
        }

        static void LoadInterfaceDLL()
        {
            // Assuming InterfaceLibrary.dll is in the same directory as the executable
            string path = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "InterfaceLibrary.dll");
            Assembly.LoadFrom(path);
        }

        static void LoadCalculationDLL()
        {
            if (calcAssembly == null)
            {
                string path = @"C:\Users\ahume\source\repos\CalculationLibrary\bin\Debug\CalculationLibrary.dll";
                calcAssembly = Assembly.LoadFrom(path);
                Type calcType = calcAssembly.GetType("CalculationLibrary.Calculator");
                calculator = (ICalculator)Activator.CreateInstance(calcType);
                Console.WriteLine("Calculation DLL loaded.");
            }
            else
            {
                Console.WriteLine("Calculation DLL is already loaded.");
```

```
        }
    }

    static void UnloadCalculationDLL()
    {
        if (calcAssembly != null)
        {
            calculator = null;
            calcAssembly = null;
            GC.Collect();
            GC.WaitForPendingFinalizers();
            Console.WriteLine("Calculation DLL unloaded.");
        }
        else
        {
            Console.WriteLine("Calculation DLL is not loaded.");
        }
    }

    static void PerformOperation(Func<double, double, double> operation)
    {
        if (calculator == null)
        {
            Console.WriteLine("Calculation DLL is not loaded. Please load it first using the 'load' command.");
            return;
        }

        try
        {
            Console.Write("Enter first number: ");
            double a = double.Parse(Console.ReadLine());
            Console.Write("Enter second number: ");
            double b = double.Parse(Console.ReadLine());
            double result = operation(a, b);
            Console.WriteLine("Result: " + result);
        }
        catch (FormatException)
        {
            Console.WriteLine("Invalid input. Please enter valid numbers.");
        }
    }
  }
}
```

Додаємо залежності, перевіряємо чи кожен з проектів правильно будується, і тоді запускаємо MainApp

Результат правильний, ми можемо повністю користуватись програмою, що складається з 3 проектів, dll яких знаходяться у різних місцях пам’яті

```
Enter a command (load, unload, add, sub, mul, div, exit):
load
Calculation DLL loaded.
Enter a command (load, unload, add, sub, mul, div, exit):
add
Enter first number: 1
Enter second number: 2
Result: 3
Enter a command (load, unload, add, sub, mul, div, exit):
sub
Enter first number: 55
Enter second number: 3
Result: 52
Enter a command (load, unload, add, sub, mul, div, exit):
```

Нічого не можна буде робити, до поки користувач не введе load, і тільки коли dll проекту з описом функцій буде скачано, можна буде користуватись калькулятором

Видалення та скачування dll проектів робить користувач введенням відповідних команд

Через різні причини, а саме низька потужність робочої станції, недосконаліть програмного забезпечення і т.д Visual Studio відмовилась запускати проект

Також було виведення графіків CPU і використання графікі

Логічно, що при запуску програми, а потім під час сортування та створення багатьох потоків збільшилось використання CPU, на графіку CPU було видно збільшення активності

А на графіку використання пам'яті було збільшення обсягу використаної пам'яті, бо кожен поток потребував виділяти для себе пам’ять, її ділянку

Також, хоч і було задіяно більше ресурсів системи, результати обчислень сортування були звісно скоріше, адже одну роботу виконувало декілька потоків

Висновок: Під час виконання лабораторної роботи, було вивчено роботу з Dll