ЛАБОРАТОРНА РОБОТА № 1

***Тема:* «Процеси та потоки**».

***Мета роботи:*** *ознайомитися з взаємодією між процесами та розподілом даних між процесами, отримати практичні навички з відображення файлу в пам’ять та синхронізації програм.*

**Хід роботи:**

***Завдання 1.*** Необхідно написати дві програми (три), які будуть мати спільні дані та одночасно до них звертатися. 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.

***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.

Структура рішення:

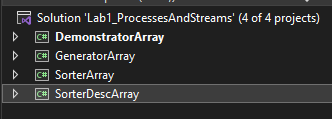


Рис. 1.1. Структура рішення

Для організації синхронізації потоків використаємо Mutex. Його ініціалізацію виконаємо у проєкті “GeneratorArray”, який відповідає за заповнння файлу числами та відображення його в пам’ять.

Лістинг методу GenerateArray у проєкті “GeneratorArray”:

private static IEnumerable<int>? GenerateArray() {

try

{

ArrayMutex.WaitOne();

var size = Random.Next(LengthLimits[0], LengthLimits[1]);

var array = new int[size];

for (var i = 0; i < size; i++)

{

array[i] = Random.Next(RangeLimits[0], RangeLimits[1]);

}

MemoryMappedFile? memoryMappedFile = null;

try

{

memoryMappedFile =

MemoryMappedFile.OpenExisting(MemoryMappedFileName, MemoryMappedFileRights.ReadWrite);

}

catch

{

memoryMappedFile ??= MemoryMappedFile.CreateFromFile(FilePath, FileMode.Create, MemoryMappedFileName,

size \* 5 + 5,

MemoryMappedFileAccess.ReadWrite);

}

using (var writer = memoryMappedFile.CreateViewAccessor(0, size \* 5 + 5))

{

writer.Write(0, size);

writer.WriteArray(5, array, 0, array.Length);

}

return array;

}

catch (Exception exception)

{

Console.WriteLine($"\nError: {exception.Message}");

return null;

}

finally

{

ArrayMutex.ReleaseMutex();

Thread.Sleep(1000);

}

}

Згенерований файл data.dat:

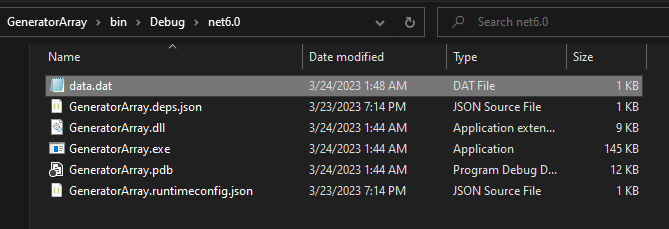


Рис. 1.2. Згенерований файл data.dat

Отже, як бачимо, першим записаним числом у файл буде розмір масиву, а вже потім всі числа масиву. Також виконується обробка виняткових ситуацій.

Розглянемо метод SortArray у проєкті “SorterArray”. Нижче наведено його лістинг:

private static void SortArray()

{

try

{

var memoryMappedFile =

MemoryMappedFile.OpenExisting(MemoryMappedFileName, MemoryMappedFileRights.ReadWrite);

var length = 2;

var savedArray = new int[1];

for (var i = 1; i < length; i++)

{

ArrayMutex = Mutex.OpenExisting(ArrayMutexName);

if (ArrayMutex == null)

{

return;

}

var mutexIsReleased = ArrayMutex.WaitOne(3000);

if (!mutexIsReleased)

{

Console.WriteLine("\nCurrently, mutex is busy with another process.");

Thread.Sleep(2000);

}

else

{

using (var reader = memoryMappedFile.CreateViewAccessor(0, 5))

{

length = reader.ReadInt32(0);

}

using (var readerWriter = memoryMappedFile.CreateViewAccessor(5, length \* 5))

{

var array = new int[length];

readerWriter.ReadArray(0, array, 0, length);

if (!array.SequenceEqual(savedArray))

{

i = 1;

}

for (var j = 0; j < length - i; j++)

{

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

{

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

}

}

readerWriter.WriteArray(0, array, 0, length);

savedArray = array;

}

Console.WriteLine($"\nThe number of permutations: {i}");

}

ArrayMutex.ReleaseMutex();

Thread.Sleep(1000);

}

}

catch (Exception exception)

{

Console.WriteLine($"\nError caught: {exception.Message}");

}

Можна помітити, що після кожного кроку сортування масиву виконується сортування виконується односекундна затримка. Алгоритм сортування — ***бульбашкове***. Важливо відмітити, що якщо поточний масив відрізняється від обробленого попереднім кроком (його змінив якийсь процес), то ми повинні розпочати сортування спочатку, тому в такому випадку ітератору присвоюється значення «1».

Третій проєкт — демонстратор поточного стану масиву. Для цього використаємо таймер на 500 мс та воркер. Замість відомбраження чисел будемо відображати відповідну кількість символів «•».

Лістинг методу StartDemo у проєкті “DemonstartorArray”:

private void StartDemo(object sender, System.ComponentModel.DoWorkEventArgs e)

{

try

{

ArrayMutex = Mutex.OpenExisting(ArrayMutexName);

if (ArrayMutex == null)

return;

var mutexIsReleased = ArrayMutex.WaitOne(3000);

if (!mutexIsReleased)

{

return;

}

memoryMappedFile = MemoryMappedFile.OpenExisting(MemoryMappedFileName, MemoryMappedFileRights.ReadWrite);

int length;

using (var reader = memoryMappedFile.CreateViewAccessor(0, 5))

{

length = reader.ReadInt32(0);

}

var array = new int[length];

using (var reader = memoryMappedFile.CreateViewAccessor(5, length \* 5))

{

reader.ReadArray(0, array, 0, length);

if (!array.SequenceEqual(\_savedArray))

{

demonstratorListBox.Items.Clear();

foreach (var item in array)

{

demonstratorListBox.Items.Add(NumberToDots(item));

}

}

\_savedArray = array;

}

}

catch

{

if (demonstratorListBox.Items[0].ToString() != "(empty)")

{

demonstratorListBox.Items.Clear();

demonstratorListBox.Items.Add("(empty)");

}

}

finally

{

ArrayMutex.ReleaseMutex();

}

}

Отже, всі критичні секції коду обробляються за допомогою try/catch і ситуація, коли некоректна робота будь-якої з трьох програм викличе неправильну роботу інших є неможливою.

Результат виконання програми:

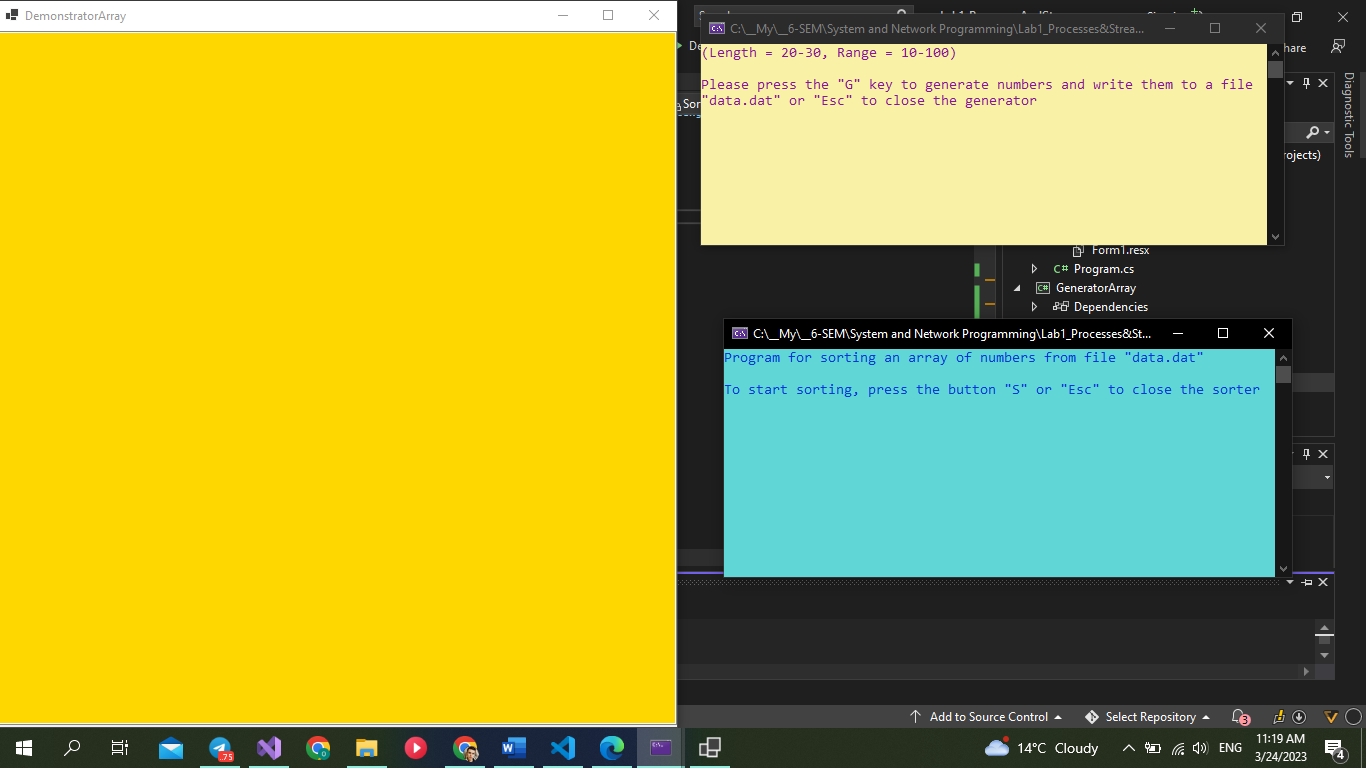


Рис. 2.1. Початково запущені програми

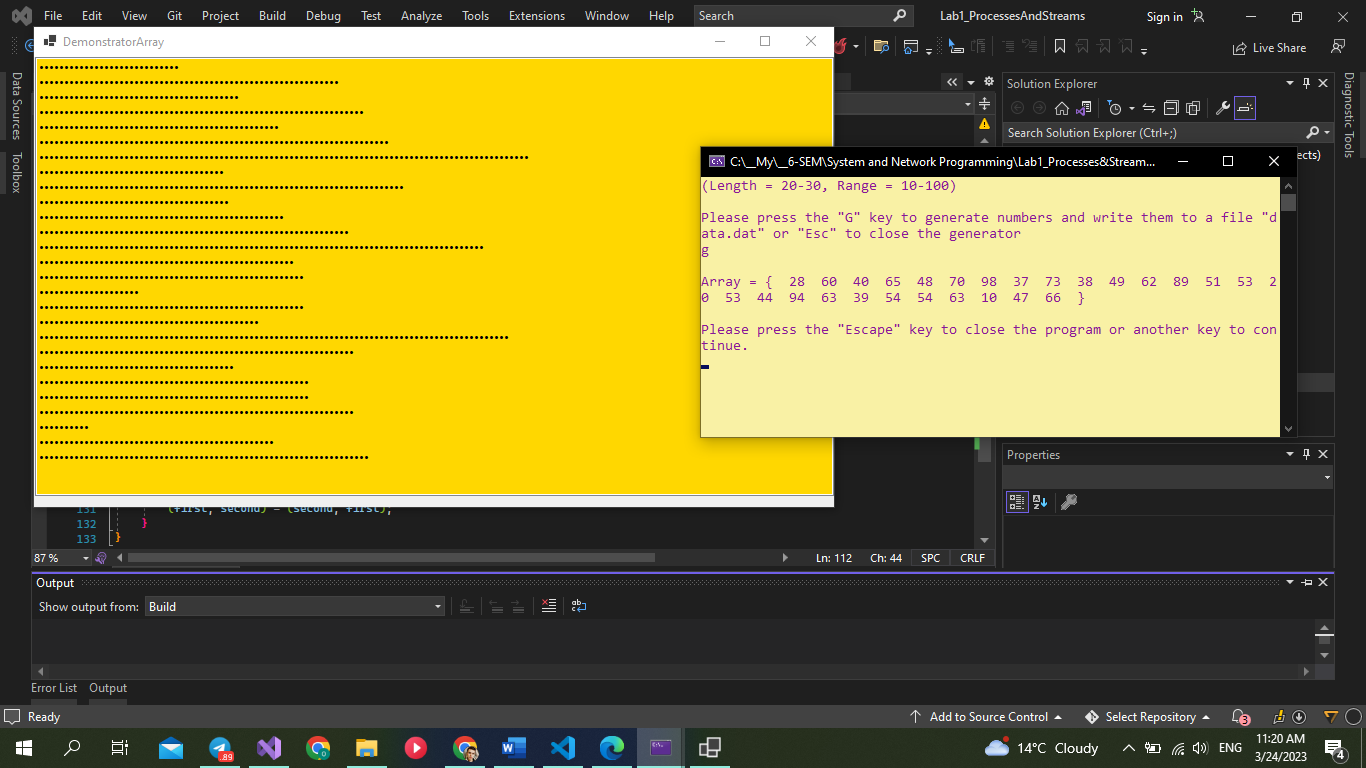
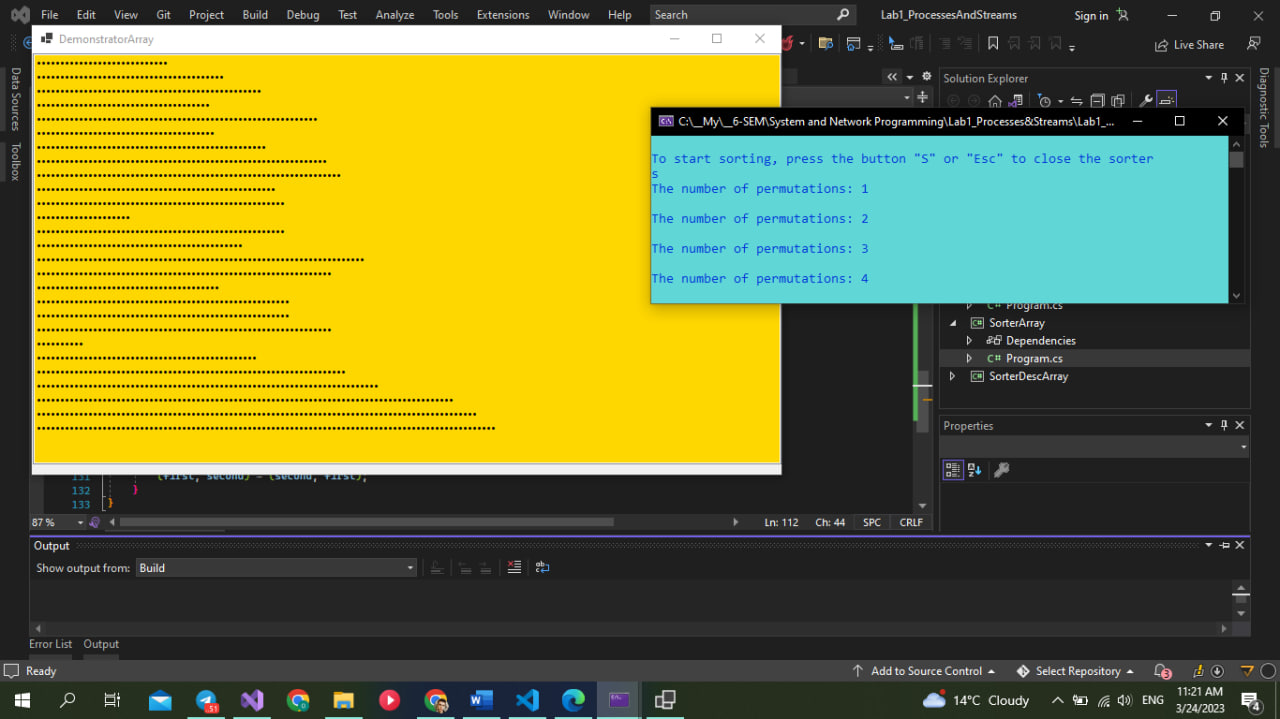
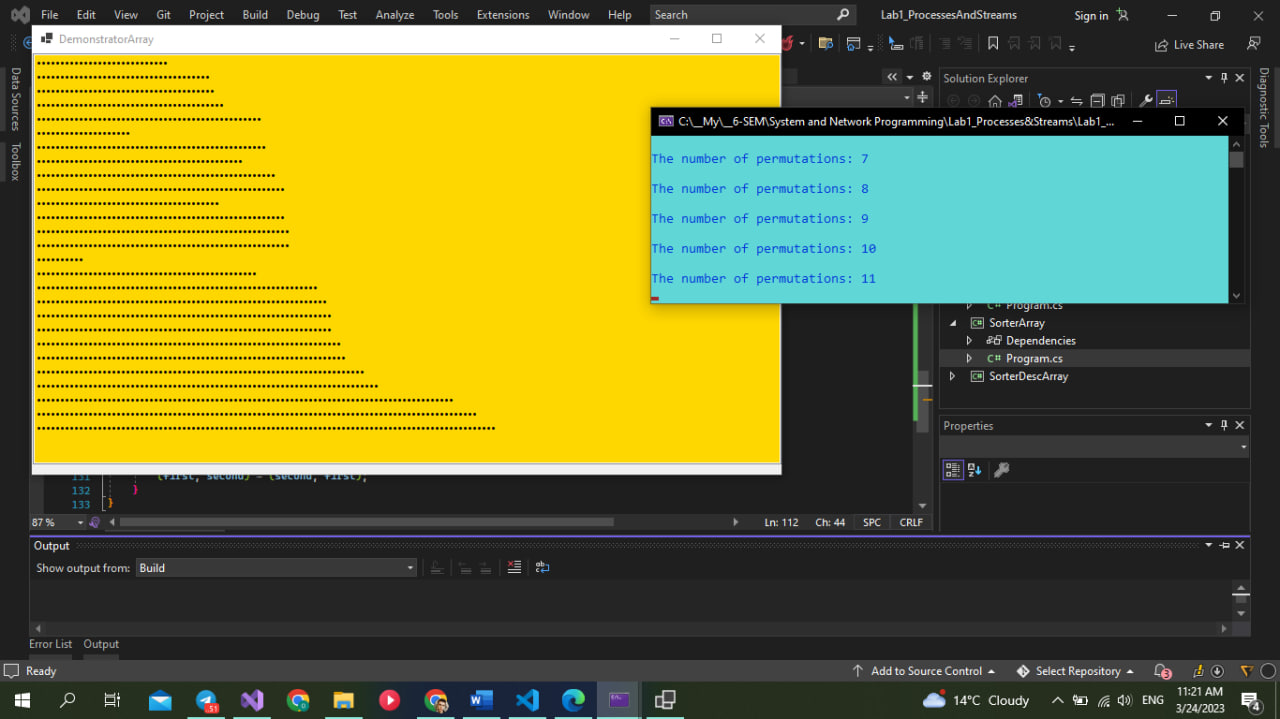
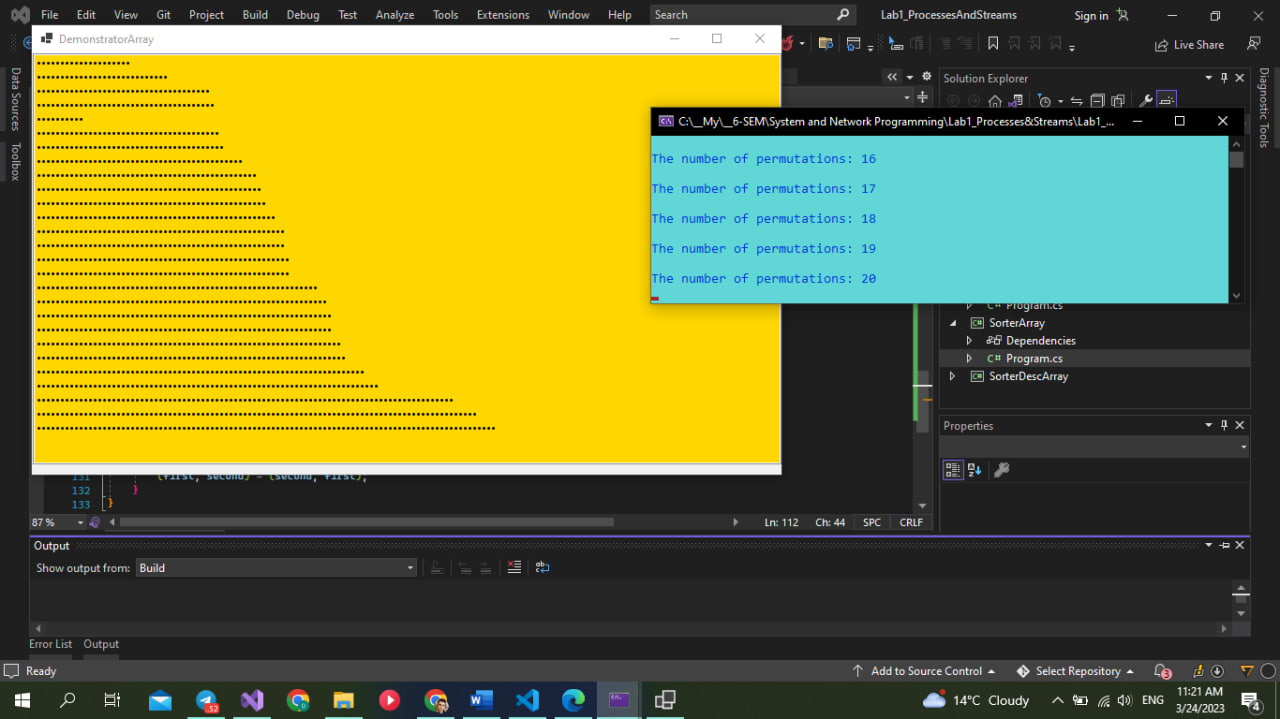


Рис. 2.1. Генерування масиву чисел та проектування файлу в пам’ять







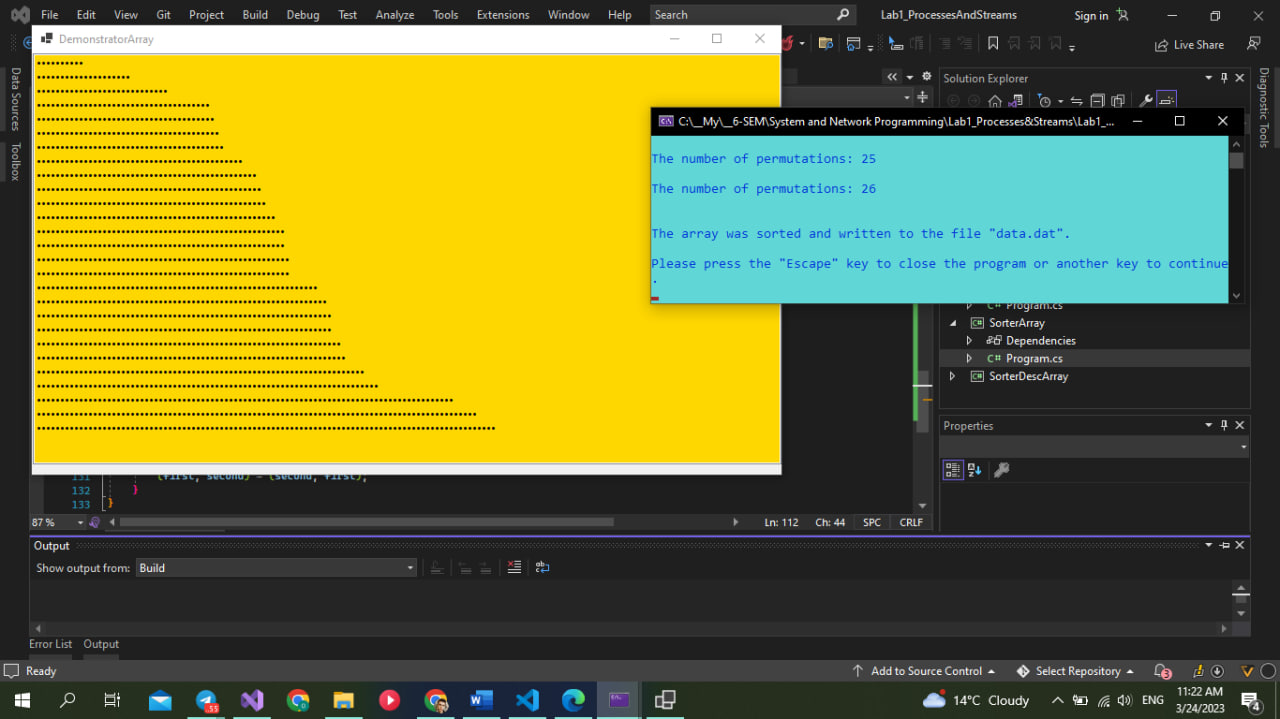


Рис. 2.2 – 2.5. Сортування бульбашкою масиву та відобрження цього на формі

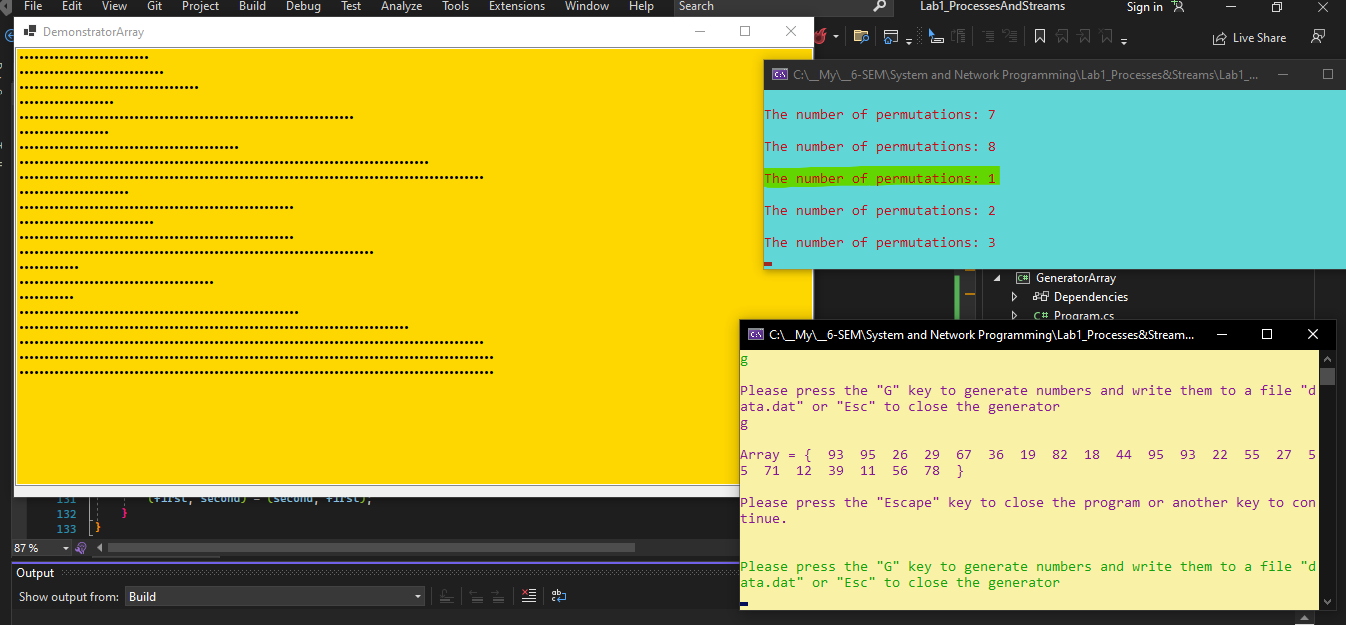


Рис. 2.6. Демонстрація одночасного генерування та сортування (сортування починається з початку)

Результат виконання програми демонструє те, що три програми виконуються по черзі, звертаючись до проєктованого файлу у пам’ять. Також з останнього скриншоту видно те, що якщо дані у файлі зміняться, то інша програма на це відреагує і таким чином робота всіх програм паралельно є коректною.

***Додаткове завдання.*** Написати четверту програму, яка буде одночасно працювати, та намагатися відсортувати той самий масив в іншому напрямку та іншим відомим методом сортування.

Реалізуємо метод сортування ***вставками за спаданням***.

Лістинг методу SortArray у проєкті “SorterArrayDesc”:

private static void SortArray()

{

try

{

var memoryMappedFile =

MemoryMappedFile.OpenExisting(MemoryMappedFileName, MemoryMappedFileRights.ReadWrite);

var length = 2;

var savedArray = new int[1];

for (var i = 1; i < length; i++)

{

ArrayMutex = Mutex.OpenExisting(ArrayMutexName);

if (ArrayMutex == null)

{

return;

}

var mutexIsReleased = ArrayMutex.WaitOne(3000);

if (!mutexIsReleased)

{

Console.WriteLine("\nCurrently, mutex is busy with another process.");

Thread.Sleep(2000);

}

else

{

using (var reader = memoryMappedFile.CreateViewAccessor(0, 5))

{

length = reader.ReadInt32(0);

}

using (var readerWriter = memoryMappedFile.CreateViewAccessor(5, length \* 5))

{

var array = new int[length];

readerWriter.ReadArray(0, array, 0, length);

if (!array.SequenceEqual(savedArray))

{

i = 1;

}

var key = array[i];

var j = i - 1;

while (j >= 0 && array[j] < key)

{

array[j + 1] = array[j];

j--;

}

array[j + 1] = key;

readerWriter.WriteArray(0, array, 0, length);

savedArray = array;

}

Console.WriteLine($"\nThe number of permutations: {i}");

}

ArrayMutex.ReleaseMutex();

Thread.Sleep(1000);

}

}

catch (Exception exception)

{

Console.WriteLine($"\nError: {exception.Message}");

}

}

Спробуємо одночасно запустити два сортувальники (з однаковою затримкою під час ітерацій — 1 секунда):

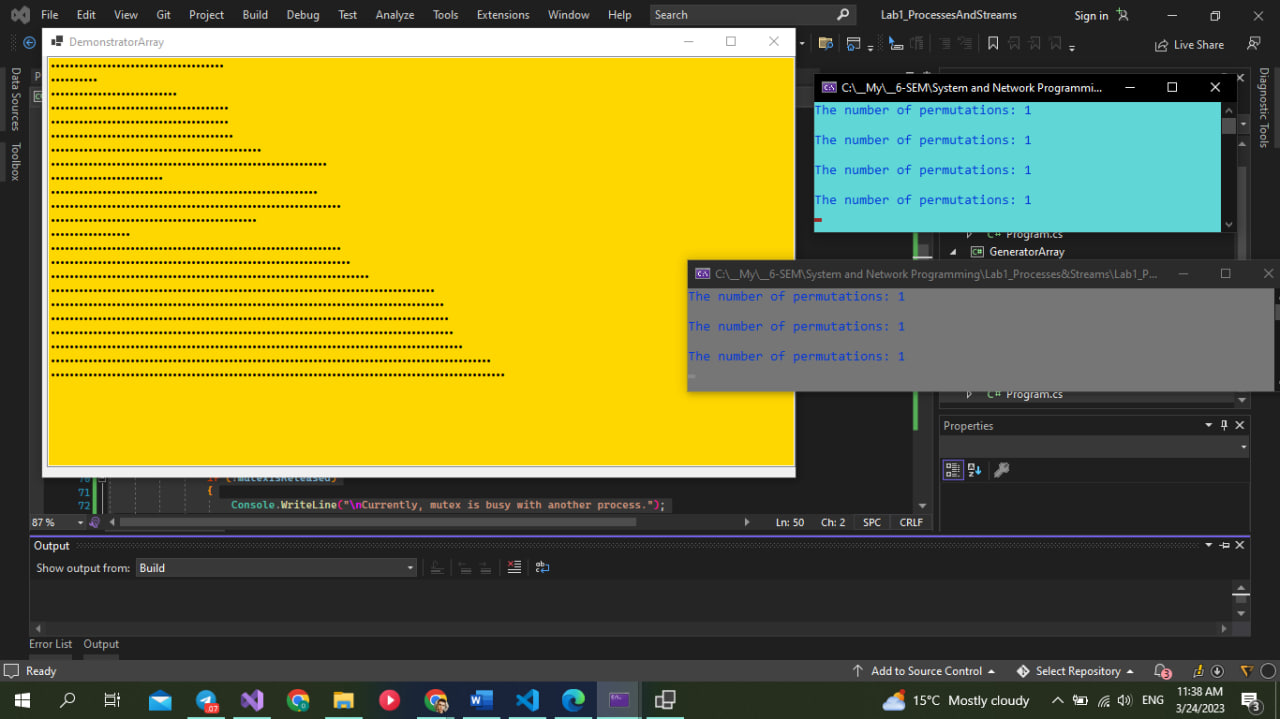


Рис. 3.1. Одночасне сортування вставками та бульбашкою в різні сторони

Але якщо закрити якусь із програм сортування, то інша програма зможе відсортувати масив. Інакше сортування триватиме нескінченно.

Якщо ми спробуємо змінити затримку під час ітерцій для сортування вставками з 1 секунди на 10, то після певного часу алгоритм бульбашкою відсортує масив за зростанням.

***Посилання на реопзиторій з вихідним кодом:***

<https://gitlab.com/2020-2024/ipz-20-2/bubenko-oleksii/system_and_network_programming/lab1_processesstreams>

***Висновок:*** у ході виконання лабораторної роботи я ознайомився із взаємодією між процесами та розподілом даних між процесами, отримав практичні навички з відображення файлу в пам’ять та синхронізації програм.