**Министерство науки и высшего образования**

**Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего профессионального образования**

**«Российский государственный университет им. А.Н. Косыгина**

**(Технологии.Дизайн.Искусство)»**

Кафедра автоматизированных систем обработки информации и управления

Лабораторная работа №4

Выполнил: Букша Кирилл Владимирович

Группа: МАГ-В-221

Вариант 4

Проверил: Кузьмина Тамара Михайловна

Москва 2021

Задание:

Создать консольное приложение, запускающее несколько потоков. Все потоки работают с двумя массивами - массивом А и массивом В, размерность у которых одинакова. Размеры массивов задаются пользователем и заполняется случайными числами. Исходные элементы массивов обязательно выводятся на экран. Массивы рассматриваются как записи векторов. Поэтому запись А + В читается, как сложение векторов, запись А\*В, определяется как покоординатное произведение векторов, 2\*А – умножение вектора на число, 2+А - прибавление числа 2 к каждому элементу массива А, т. е. программа выполняет действия над векторами и числами и получает в результате вектор. Каждое действие оформить как отдельную функцию.

Выполнить синхронизацию потоков, определенных в 3 лабораторной работе, двумя разными способами.

Нужно использовать оператор lock, классы Monitor, Mutex.

Вариант: (А + 7 \* В) \* 3

Решение:

Были скопированы базовые функции для обработки вводимой информации, генерации и вывода данных:

private static (int[], int[]) GenerateData(int size)

{

var rnd = new Random();

var vectorA = Enumerable.Range(0, size).Select(\_ => rnd.Next(1, 20)).ToArray();

var vectorB = Enumerable.Range(0, size).Select(\_ => rnd.Next(1, 20)).ToArray();

return (vectorA, vectorB);

}

private static int ReadUserInput()

{

while (true)

{

Console.WriteLine("Enter q to exit.");

Console.Write("Enter size of vectors: ");

var input = Console.ReadLine();

if (int.TryParse(input, out var size))

{

if (size <= 0)

{

Console.WriteLine("Size must be positive integer.");

continue;

}

return size;

}

if (input == "q")

throw new Exception();

}

}

private static void PrintData(IEnumerable<int> enumerable, string header)

{

Console.Write($"{header}: ");

foreach (var item in enumerable) Console.Write($"{item} ");

Console.WriteLine();

}

Также были скопированы методы, реализующие базовые операции над векторами. Были реализованы новые методы использующие синхронизацию потоков:

private static int[] Sum(int[] first, int[] second)

{

if (first.Length != second.Length)

throw new ArgumentException("Pass lists with the same size!");

var zippedLists = first.Zip(second, (f, s) => new { First = f, Second = s });

return zippedLists.Select(item => item.First + item.Second).ToArray();

}

private static int[] Multiply(int[] vector, int k)

{

return vector.Select(n => n \* k).ToArray();

}

private static void MultiplyAndAddResult(int[] vector, int k, IThreadSafeArray result)

{

var temporaryResult = vector.Select(n => n \* k).ToList();

for (var index = 0; index < temporaryResult.Count; index++)

{

result.AddToValueByIndex(index, temporaryResult[index]);

}

}

Следующим шагом стала разработка тестируемых функций. Реализовано три варианта: mutex, monitor, а также скопирована реализация из предыдущей лабораторной работы для сравнения. Код функций представлен ниже:

private static int[] CalculateFunctionMultipleThreadSmart(int[] vectorA, int[] vectorB)

{

var firstMultiplyTask = Task.Run(() => Multiply(vectorB, 21));

var secondMultiplyTask = Task.Run(() => Multiply(vectorA, 3));

Task.WaitAll(firstMultiplyTask, secondMultiplyTask);

var firstMultiplyResult = firstMultiplyTask.Result;

var secondMultiplyResult = secondMultiplyTask.Result;

var finalResult = Task.Run(() => Sum(firstMultiplyResult, secondMultiplyResult)).Result;

return finalResult;

}

private static int[] CalculateFunctionMutex(int[] vectorA, int[] vectorB)

{

// Check input length

var lengthA = vectorA.Length;

var lengthB = vectorB.Length;

if (lengthA != lengthB)

throw new ArgumentException("Input vectors must have same size!");

// Initializing shared resource

var result = new MutexArray(lengthA);

// Starting two threads

var firstMultiplyTask = Task.Run(() => MultiplyAndAddResult(vectorB, 21, result));

var secondMultiplyTask = Task.Run(() => MultiplyAndAddResult(vectorA, 3, result));

// Waiting until task is finished

Task.WaitAll(firstMultiplyTask, secondMultiplyTask);

return result.GetData();

}

private static int[] CalculateFunctionMonitor(int[] vectorA, int[] vectorB)

{

// Check input length

var lengthA = vectorA.Length;

var lengthB = vectorB.Length;

if (lengthA != lengthB)

throw new ArgumentException("Input vectors must have same size!");

// Initializing shared resource

var result = new MonitorArray(lengthA);

// Starting two threads

var firstMultiplyTask = Task.Run(() => MultiplyAndAddResult(vectorB, 21, result));

var secondMultiplyTask = Task.Run(() => MultiplyAndAddResult(vectorA, 3, result));

// Waiting until task is finished

Task.WaitAll(firstMultiplyTask, secondMultiplyTask);

return result.GetData();

}

Для удобства были написаны вспомогательные классы, которые реализуют описанный интерфейс и используют внутри себя mutex или monitor для разграничения доступа к массиву данных:

public class MutexArray : IThreadSafeArray

{

private readonly Mutex \_mutex = new();

private readonly int[] \_data;

public MutexArray(int size)

{

\_data = new int[size];

}

public MutexArray(IEnumerable<int> initialData)

{

\_data = initialData.ToArray();

}

public int[] GetData()

{

\_mutex.WaitOne();

try

{

var result = \_data.ToArray();

return result;

}

finally

{

\_mutex.ReleaseMutex();

}

}

public void AddToValueByIndex(int index, int value)

{

#region Educational information

//var id = new Random().Next(int.MaxValue);

//Console.WriteLine($"AddToValueByIndex: {id} is waiting for mutex release.");

#endregion

\_mutex.WaitOne();

try

{

#region Educational information

//Console.WriteLine($"AddToValueByIndex: {id} occupied mutex.");

#endregion

\_data[index] += value;

}

finally

{

\_mutex.ReleaseMutex();

#region Educational information

//Console.WriteLine($"AddToValueByIndex: {id} released mutex.");

#endregion

}

}

}

public class MonitorArray : IThreadSafeArray

{

private readonly object \_locker = new object();

private readonly int[] \_data;

public MonitorArray(int size)

{

\_data = new int[size];

}

public MonitorArray(IEnumerable<int> initialData)

{

\_data = initialData.ToArray();

}

public int[] GetData()

{

Monitor.Enter(\_locker);

try

{

var result = \_data.ToArray();

return result;

}

finally

{

Monitor.Exit(\_locker);

}

}

public void AddToValueByIndex(int index, int value)

{

#region Educational information

//var id = new Random().Next(int.MaxValue);

//Console.WriteLine($"AddToValueByIndex: {id} is waiting for monitor release.");

#endregion

Monitor.Enter(\_locker);

try

{

#region Educational information

//Console.WriteLine($"AddToValueByIndex: {id} occupied monitor.");

#endregion

\_data[index] += value;

}

finally

{

Monitor.Exit(\_locker);

#region Educational information

//Console.WriteLine($"AddToValueByIndex: {id} released monitor.");

#endregion

}

}

}

public interface IThreadSafeArray

{

public void AddToValueByIndex(int index, int value);

}

Финальным шагом стала разработка функции Main которая использует описанные ранее методы и классы:

private static void Main()

{

int size;

try

{

size = ReadUserInput();

}

catch

{

return;

}

var (vectorA, vectorB) = GenerateData(size);

PrintData(vectorA, nameof(vectorA));

PrintData(vectorB, nameof(vectorB));

// Multiple Thread Smart

Console.WriteLine("(From Lab 3) Multiple Thread Smart results.");

var watch = Stopwatch.StartNew();

var result = CalculateFunctionMultipleThreadSmart(vectorA, vectorB);

watch.Stop();

PrintData(result, nameof(result));

Console.WriteLine($"Multiple thread Smart, time of execution {watch.ElapsedMilliseconds} ms");

Console.WriteLine();

// Concurrent access with Mutex

Console.WriteLine("Mutex.");

watch = Stopwatch.StartNew();

result = CalculateFunctionMutex(vectorA, vectorB);

watch.Stop();

PrintData(result, nameof(result));

Console.WriteLine($"Mutex, time of execution {watch.ElapsedMilliseconds} ms");

Console.WriteLine();

// Concurrent access with Mutex

Console.WriteLine("Monitor.");

watch = Stopwatch.StartNew();

result = CalculateFunctionMonitor(vectorA, vectorB);

watch.Stop();

PrintData(result, nameof(result));

Console.WriteLine($"Monitor, time of execution {watch.ElapsedMilliseconds} ms");

Console.WriteLine();

}

После запуска программы появляются следующие результаты:

Enter q to exit.

Enter size of vectors: 10

vectorA: 9 16 12 12 16 13 6 4 9 10

vectorB: 15 18 12 5 16 10 12 2 15 10

(From Lab 3) Multiple Thread Smart results.

result: 342 426 288 141 384 249 270 54 342 240

Multiple thread Smart, time of execution 60 ms

Mutex.

result: 342 426 288 141 384 249 270 54 342 240

Mutex, time of execution 4 ms

Monitor.

result: 342 426 288 141 384 249 270 54 342 240

Monitor, time of execution 1 ms

При работе с массивом большей размерности разница между Mutex и Monitor становится более заметной и достигает разрыва в 50 и более раз. Mutex является более «тяжелым» классом и рекомендуется к использованию только в продвинутых сценариях, когда необходима работа сразу в нескольких процессах.

Вывод: была написана программа, использующая многопоточный подход к программированию, а также классы Monitor и Mutex. Программа позволяет сравнить различные подходы к решению задачи и сделать выводы. Программа отлажена и протестирована ручными методами тестирования. Исходный код программы залит на Github и доступен по ссылке: https://github.com/bukSHA1024/RSU\_TRPO\_Lab4