Introduction

1. Download and unzip 01.Multithreading.v.1.2.zip archive. Restore packages via “dotnet restore” if needed.
2. Implement the tasks 1-6 using templates from the Multithreading.sln.

Task 1 (MultiThreading.Task1.100Tasks.csproj)

Write a program, which creates an array of 100 Tasks, runs them and waits all of them are not finished. Each Task should iterate from 1 to 1000 and print into the console the following string:

“Task #0 – {iteration number}”.

Task 2 (MultiThreading.Task2.Chaining.csproj)

Write a program, which creates a chain of four Tasks.

1. First Task – creates an array of 10 random integer.
2. Second Task – multiplies this array with another random integer.
3. Third Task – sorts this array by ascending.
4. Fourth Task – calculates the average value.

All these tasks should print the values to console.

Task 3 (MultiThreading.Task3.MatrixMultiplier.csproj, MultiThreading.Task3.MatrixMultiplier.Tests.csproj)

Write a program, which multiplies two matrices and uses class Parallel.

Details

Open *MultiThreading.Task3.MatrixMultiplier.csproj*.

a. Implement logic of MatricesMultiplierParallel.cs using Parallel class.

Make sure that all the tests within MultiThreading.Task3.MatrixMultiplier.Tests.csproj run successfully.

b. Create a test inside MultiThreading.Task3.MatrixMultiplier.Tests.csproj to check which multiplier runs faster. Find out the size which makes parallel multiplication more effective than the regular one.

Task 4 (MultiThreading.Task4.Threads.Join.csproj)

Write a program which recursively creates 10 threads.

Each thread should be with the same body and receive a state with integer number, decrement it, print and pass as a state into the newly created thread.

Use Thread class for this task and Join for waiting threads.

Implement all the following options:

- a) Use Thread class for this task and Join for waiting threads.
- b) ThreadPool class for this task and Semaphore for waiting threads.

Task 5 (MultiThreading.Task5.Threads.SharedCollection.csproj)

Write a program which creates two threads and a shared collection:

the first one should add 10 elements into the collection and the second should print all elements in the collection after each adding (in other words, if the first list contains elements from 1 to 10, the second thread should print something like [1], [1, 2], [1, 2, 3], …, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], the amount of elements should increase).

Use Thread, ThreadPool or Task classes for thread creation and any kind of synchronization constructions.

Task 6 (MultiThreading.Task6.Continuation.csproj)

Create a Task and attach continuations to it according to the following criteria:

a. Continuation task should be executed regardless of the result of the parent task.

b. Continuation task should be executed when the parent task finished without success.

c. Continuation task should be executed when the parent task would be finished with fail and parent task thread should be reused for continuation

d. Continuation task should be executed outside of the thread pool when the parent task would be cancelled

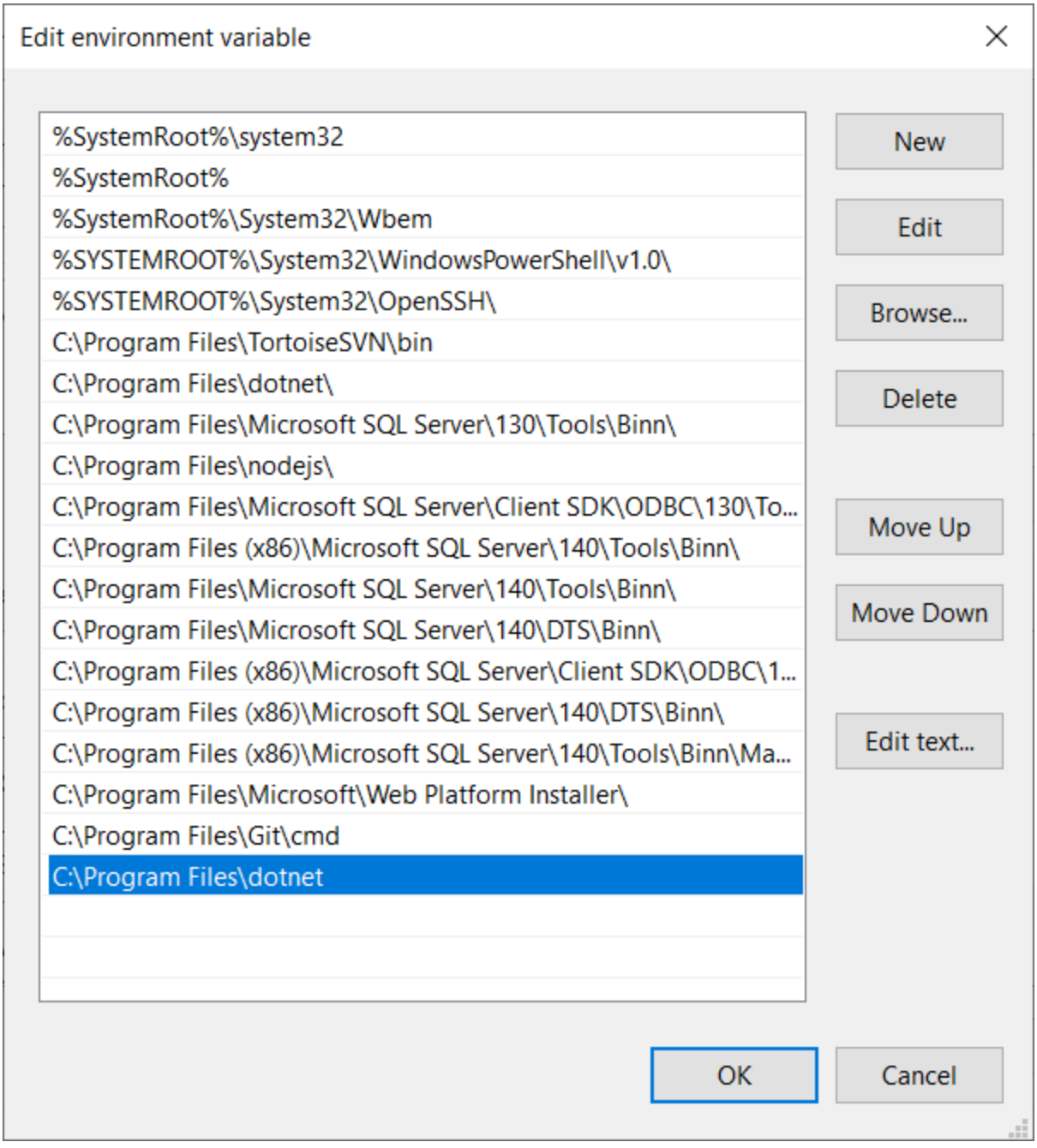
Demonstrate the work of each case with console utility.

Task 7. Multi-threading theory.

Please learn the Theory to prepare for the meeting with your mentor.

Notes

If “dotnet restore” does not work:



Please make sure that Path variable contains link to dotnet folder.

Questions for check:

1. **Активные (foreground) and фоновые (background) потоки**.

- В чем отличие? К какому виду потоков по умолчанию относятся:
    - потоки, созданные при конструировании объекта Thread;
    - потоки в ThreadPool;
    - потоки управляемой среды, созданные неуправляемым кодом?
- Приведите примеры, когда следует использовать foreground-, а когда background потоки?
- Какие средства предоставляет .Net Framework для создания background/foreground потоков?

  Джефри Рихтер. Глава 26. “Фоновые и активные потоки”

2. **Конструкции синхронихации (locking)**

- Что такое Mutex? Отличие от других конструкций синхронизации.
- Отличие между lock и Monitor.
- Как много времени затрачивают lock-конструкции на служебные операции (overhead)? Когда следует их использовать (примеры)?
- Как не допустить deadlock при использовании конструкций блокировки? Опишите, что может вызвать deadlock в multithreading. Напишите пример кода, проверьте его в Visual Studio (не следует путать с deadlock в async/await).

  Joseph Albahari "Locking"

3. **Blocking vs Spinning**

- Что такое spinning и его отличие от блокировки?

- SpinLock and SpinWait - для чего нужны, привести примеры, где они могут быть эффективны.

  Joseph Albahari "Blocking vs Spinning", “SpinLock and SpinWait”.

4. **Thread safety**

- Какие способы используются для написания потокобезопасного кода?
- Как реализовать потокобезопасный Singleton?

  Joseph Albahari "Thread safety"

  “Thread safe singleton” Design Pattern.

5. **Event wait handles**

- Что такое signaling в многопоточности?
- Отличие между WaitAny, WaitAll and SignalAndWait.
- ManualResetEvent vs AutoResetEvent.
- Как много времени затрачивают signaling-конструкции на служебные операции (overhead)?
- Какие существуют механизмы для отправки сигналов между процессами (cross-process signaling)?

  Joseph Albahari "Signaling with Event Wait Handles"

6. **Concurrent collections**

- Назовите разновидности ConcurrentCollections, в каких случаях следует применять тот или иной вид коллекции?

  Microsoft Docs "System.Collections.Concurrent Namespace".

  Joseph Albahari “Concurrent Collections”

7. **Parallelism**

- Что такое параллелизм? Какие средства предоставляет .Net Framework для параллельного выполнения кода?
- Может ли параллельное исполнение кода быть менее эффективным, чем последовательное? Приведите примеры в подтверждение.
- Какую роль играет AggregateException в Parallel Programming?
-