Concurrent Collections Documentation

Contents

[Overview 2](#_Toc1062577)

[General Notes For All Processes 2](#_Toc1062578)

[Parallel.Invoke(Action[] actions) { } 2](#_Toc1062579)

# Overview

Our examples in this solution are using primitive data types like regular values and objects. We are going to be pushing user defined objects and list onto the queues and stacks. So, many of these notes are going to do if we had files with actual records.

# General Notes For All Processes

Using Task for Parallel Processing is completely different than using Task for asynchronous processing.

# Parallel.Invoke(Action[] actions) { }

The Parallel.Invoke(Action [] actions) method allows us to run an array of action delegates. We can use these action delegates to run the same method over and over again concurrently or we can run different methods concurrently. (There is no guarantee that these methods will run concurrently.) One example we have of this is the Queues class of this library.

There are several different ways of implementing this.

Option 1

1. Build our Methods that we want to run in parallel
2. Call them in an action delegate
3. Pass them as parameters in the Parallel.Invoke method.

Option 2

1. Write an inline Action delegate using a lambda function

# Exception Handling

We want to put the exception handling as close to the code that is processing as possible. This is because if we try to encapsulate the whole code block in a try catch phrase, our code by continue to execute when we want to try to catch and handle an exception. Remember the whole point of this is multi-threading.

1. Pas the action into the Parallel.Invoke method several times.  
   A screenshot of a cell phone

   Description automatically generated

# PLINQ (Parallel Language Integrated Query)

* PLINQ is what we are going to use for processing data in Parallel.
* This works by partition the data source into segments and then executing the query on each segment on separate worker threads in parallel on multiple processors.
  + In many cases, parallel execution means that the query runs significantly faster.
* We use this to write pull-based calculations over sequences.

PLINQ Screenshot Example #1

A screenshot of a cell phone

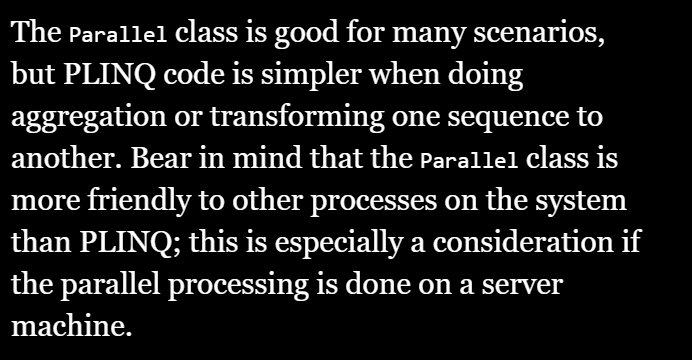
Description automatically generated

PLINQ Screenshot Example #2

A picture containing animal

Description automatically generated

# Gotchas

* PLINQ Queries use all of the cores on a machine. So, the following rule of thumb from our Concurrent text book is that
  + 
* It is easy to have race conditions when using static variables or shared variables.
* Parallelism is not always faster. If there is a question of performance or speed use the System.Diagonistics.Stopwatch to answer it
* Avoid calls to non-thread-safe methods.

Thread Safe Conversions

|  |  |  |
| --- | --- | --- |
| Not Thread Safe | Possible Replacement | Notes |
| List<T> | ConcurrentBag<T> | We can use the ConcurrentBag in a Parallel Method and then convert that to a list. |
| List<T> | Concurrent<T> | We could use a autoincremented |
| Queue<T> | ConccurrentQueue<T> |  |
| Stack<T> | ConcurrentStack<T> |  |
|  |  |  |