Starting Tasks In foreach Loop Uses Value of Last Item [duplicate]

Asked 13 years, 11 months ago Modified 3 years, 2 months ago Viewed 32k times

54

This question already has answers here:

Captured variable in a loop in C# (11 answers)

Closed 5 years ago.

I am making a first attempt at playing with the new Tasks, but something is happening that I don't understand.

First, the code, which is pretty straight-forward. I pass in a list of paths to some image files, and attempt to add a task to process each of them:

```
public Boolean AddPictures(IList<string> paths)
{
    Boolean result = (paths.Count > 0);
    List<Task> tasks = new List<Task>(paths.Count);

    foreach (string path in paths)
    {
        var task = Task.Factory.StartNew(() =>
            {
                Boolean taskResult = ProcessPicture(path);
                return taskResult;
            });
        task.ContinueWith(t => result &= t.Result);
        tasks.Add(task);
    }

    Task.WaitAll(tasks.ToArray());

    return result;
}
```

I've found that if I just let this run with, say, a list of 3 paths in a unit test, all three tasks use the last path in the provided list. If I step through (and slow down the processing of the loop), each path from the loop is used.

Can somebody please explain what is happening, and why? Possible workarounds?

c# multithreading task-parallel-library

Share

Improve this question

Follow

edited Mar 14, 2017 at 2:02

abatishchev

100k ● 88 ● 301 ● 442

asked Jan 13, 2011 at 19:27

Wonko the Sane

10.8k ● 8 ● 69 ● 95

3 May I suggest using ReSharper. This particular error and other potential bugs are highlighten for you – Rune FS Jan 13, 2011 at 19:53

2 Answers

Sorted by: Highest score (default)

102

You're closing over the loop variable. Don't do that. Take a copy instead:

```
foreach (string path in paths)
{
    string pathCopy = path;
    var task = Task.Factory.StartNew(() =>
        {
            Boolean taskResult = ProcessPicture(pathCopy);
            return taskResult;
        });
    // See note at end of post
    task.ContinueWith(t => result &= t.Result);
    tasks.Add(task);
}
```

Your current code is capturing `path` - not the *value* of it when you create the task, but the variable itself. That variable changes value each time you go through the loop - so it can easily change by the time your delegate is called.

By taking a copy of the variable, you're introducing a *new* variable each time you go through the loop - when you capture *that* variable, it won't be changed in the next iteration of the loop.

Eric Lippert has a pair of blog posts which go into this in a lot more detail: part 1; part 2.

Don't feel bad - this catches almost everyone out :(

Note about this line:

```
task.ContinueWith(t => result &= t.Result);
```

As pointed out in comments, this isn't thread-safe. Multiple threads could execute it at the same time, potentially stamping on each other's results. I haven't added locking or anything similar as it would distract from the main issue that the question is interested, namely variable capture. However, it's worth being aware of.

Share

Improve this answer

Follow

edited Oct 8, 2021 at 9:26

formicini

348 ● 5 ● 17

answered Jan 13, 2011 at 19:29

Jon Skeet

1.5m ● 889 ● 9.3k ● 9.3k

1 But of course. Forest for the trees and all that. :) – Wonko the Sane Jan 13, 2011 at 19:32

1 this closure problem and in-proper use of Random() must be in the top 5 frequency-wise at SO – BrokenGlass Jan 13, 2011 at 19:35

2 This behaviour has changed, not only in C#5.0 (as noted in the update on Eric Lipperts blog article), but also in VS2012 if you're targeting 4.0. – Snixtor Mar 1, 2013 at 0:42

5 @Snixtor: That's still the C# 5 compiler. It's important to distinguish between the *language* version you're using, and the *framework* version you're targeting. – Jon Skeet Mar 1, 2013 at 6:42

2 This code has a race condition in that you're aggregating the results into the `result` variable from multiple threads without proper synchronization. – Servy Nov 14, 2018 at 15:49

12

The lambda that you're passing to `StartNew` is referencing the `path` variable, which changes on each iteration (i.e. your lambda is making use of the *reference* of `path`, rather than just its value). You can create a local copy of it so that you aren't pointing to a version that will change:

```
foreach (string path in paths)
{
    var lambdaPath = path;
    var task = Task.Factory.StartNew(() =>
        {
            Boolean taskResult = ProcessPicture(lambdaPath);
            return taskResult;
        });
    task.ContinueWith(t => result &= t.Result);
    tasks.Add(task);
}
```

Share Improve this answer Follow

answered Jan 13, 2011 at 19:29

bdukes

156k ● 25 ● 150 ● 176