When should I not use the ThreadPool in .Net? [closed]

Asked 16 years, 4 months ago Modified 6 years, 5 months ago

Viewed 18k times

39

Closed. This question needs to be more focused. It is not currently accepting answers.

Want to improve this question? Update the question so it focuses on one problem only by editing this post.

Closed 8 years ago.

Improve this question

When should I **not** use the ThreadPool in .Net?

It looks like the best option is to use a ThreadPool, in which case, why is it not the only option?

What are your experiences around this?

c# .net multithreading design-decisions

Share

Improve this question

edited Apr 15, 2013 at 12:56

Gennady Vanin
Геннадий Ванин

Follow

10.4k ● 12 ● 80 ● 109

asked Aug 13, 2008 at 19:24

Vaibhav

11.4k ● 11 ● 53 ● 71

9 Answers

Sorted by: Highest score (default)

30

@Eric, I'm going to have to agree with Dean. Threads are expensive. You can't assume that your program is the only one running. When everyone is greedy with resources, the problem multiplies.

> I prefer to create my threads manually and control them myself. It keeps the code very easy to understand.

That's fine when it's appropriate. If you need a bunch of worker threads, though, all you've done is make your code more complicated. Now you have to write code to manage them. If you just used a thread pool, you'd get all the thread management for free. And the thread pool provided by the language is very likely to be more robust, more efficient, and less buggy than whatever you roll for yourself.

> ```
> Thread t = new Thread(new ThreadStart(DoSomething)
> t.Start();
> t.Join();
> ```

I hope that you would normally have some additional code in between `Start()` and `Join()`. Otherwise, the extra thread is useless, and you're wasting resources for no reason.

> People are way too afraid of the resources used by threads. I've never seen creating and starting a thread to take more than a millisecond. There is no hard limit on the number of threads you can create. RAM usage is minimal. Once you have a few hundred threads, CPU becomes an issue because of context switches, so at that point you might want to get fancy with your design.

A millisecond is a *long* time on modern hardware. That's 3 million cycles on a 3GHz machine. And again, you aren't the only one creating threads. Your threads compete for the CPU along with every other program's threads. If you use not-quite-too-many threads, and so does another program, then together you've used too many threads.

> Seriously, don't make life more complex than it needs to be. Don't use the thread pool unless you need something very specific that it offers.

Indeed. Don't make life more complex. If your program needs multiple worker threads, don't reinvent the wheel. Use the thread pool. That's why it's there. Would you roll your own string class?

Share Improve this answer

Follow

edited Oct 25, 2013 at 18:53

Ian Boyd

256k ● 264 ● 907 ● 1.3k

answered Aug 13, 2008 at 21:05

Derek Park

46.8k ● 16 ● 59 ● 76

3 Some people rolls their own wrapper over an array of char's and use it like a string.. so it's possible... and it's sad too. – Andrei Rînea Oct 9, 2008 at 11:48

1 I down voted this. This is isn't an answer; it's a reply to someone else's. – hwiechers Aug 13, 2012 at 4:49

Tis well enough an answer. But 22 upvotes, at this moment? Its not *that* good. There isn't a huge votering being run out of Redmond, is there? – user1228 Aug 22, 2012 at 12:59

@Will, it's a 4-year old comment. It's averaging less than half an upvote per month. Hardly an unbelievable level of votes. – Derek Park Sep 21, 2012 at 15:19

2 @hwiechers, it's half answer, half reply. SO didn't have comments when this was posted, and I'm not willing to waste the time to go through all my old answers and convert them to comments. – Derek Park Sep 21, 2012 at 15:22

18

The only reason why I wouldn't use the `ThreadPool` for cheap multithreading is if I need to…

1. interract with the method running (e.g., to kill it)
2. run code on a STA thread (this happened to me)

3. keep the thread alive after my application has died (`ThreadPool` threads are background threads)
4. in case I need to change the priority of the Thread. We can not change priority of threads in ThreadPool which is by default Normal.

> **P.S.:** The MSDN article *"The Managed Thread Pool"* contains a section titled, *"When Not to Use Thread Pool Threads"*, with a very similar but slightly more complete list of possible reasons for not using the thread pool.

There are lots of reasons why you would need to skip the `ThreadPool`, but if you don't know them then the `ThreadPool` should be good enough for you.

Alternatively, look at the new Parallel Extensions Framework, which has some neat stuff in there that may suit your needs without having to use the `ThreadPool`.

Share Improve this answer Follow

edited Jul 12, 2018 at 14:55

V0d01ey

49 ● 2 ● 6

answered Aug 13, 2008 at 19:32

user1228

u said :- keep the thread alive after my application has died (ThreadPool threads are background threads) but how far i know background thread always end with main thread. so what do u say?? – Mou Apr 29, 2013 at 20:05

2 @Mou: I say that I have no idea what you're trying to say, unfortunately. – user1228 Apr 29, 2013 at 20:37

If you're reading this, then you should be using Tasks instead of the ThreadPool now. – user1228 Apr 21, 2015 at 15:47

u said `keep the thread alive after my application has died (ThreadPool threads are background threads)` ... am i right. is it true ? if thread pool thread is background thread then thread can run when main thread died ? explain plzz – Mou Apr 21, 2015 at 18:56

1 I've run across issues #1 (adding CancellationToken support to a long-running (5+ minute) "pure" (enough) function call in a third-party library) and #2 (concurrent WPF ImageSource rendering... each concurrent render needs to start and end on the same STA thread, but then can be frozen to be displayed from other threads) in the same application, so this is the right answer for me. Most of the time, the managed thread pool (or one of the zillion other things like TPL that ultimately use it) is correct, but sometimes the right answer really is to spin up a new Thread and move on with your life. – Joe Amenta Jun 13, 2015 at 12:31

10

To quarrelsome's answer, I would add that it's best not to use a ThreadPool thread if you need to guarantee that your thread will begin work immediately. The maximum number of running thread-pooled threads is limited per appdomain, so your piece of work may have to wait if

they're all busy. It's called "queue user work item", after all.

Two caveats, of course:

1. You can change the maximum number of thread-pooled threads in code, at runtime, so there's nothing to stop you checking the current vs maximum number and upping the maximum if required.
2. Spinning up a new thread comes with its own time penalty - whether it's worthwhile for you to take the hit depends on your circumstances.

Share Improve this answer

Follow

answered Aug 13, 2008 at 19:34

Dogmang

727 ● 1 ● 7 ● 14

This is the correct answer. The number of times I have seen long delays while the hill climbing alg in the OS decides to create a new thread to clear that ever increasing item queue would shock you (I have seen delays of over 4 seconds before). MS state very clearly that the threadpool is meant for background tasks that are NOT long running, in fact they have a flag called "LongRunning" which you can set for the thread via the threadpool classes that actually creates that thread outside of the managed pool e.g. a a new separate thread and unmanaged for that very reason. – Skyline Jan 19, 2022 at 16:45

Thread pools make sense whenever you have the concept of worker threads. Any time you can easily partition processing into smaller jobs, each of which can

9

be processed independently, worker threads (and therefore a thread pool) make sense.

Thread pools do not make sense when you need thread which perform entirely dissimilar and unrelated actions, which cannot be considered "jobs"; e.g., One thread for GUI event handling, another for backend processing. Thread pools also don't make sense when processing forms a pipeline.

Basically, if you have threads which start, process a job, and quit, a thread pool is probably the way to go. Otherwise, the thread pool isn't really going to help.

Share Improve this answer

Follow

answered Aug 13, 2008 at 19:32

Derek Park

46.8k ● 16 ● 59 ● 76

2 Can you explain the remark "thread pools don't make sense when processing forms a pipeline"? Suppose I have work items that need to be compressed, then encrypted, and compression uses 10x the compute as encryption. Why not use a threadpool with a 10:1 ratio of compressor to encryptor threads? – Cheeso Mar 26, 2009 at 19:21

2 Thread pools are generally for when a program has independent, discrete pieces of work to do. If there's communication between the worker threads (such as in a pipeline), then you don't really have a thread pool scenario. – Derek Park Apr 3, 2009 at 2:21

> I'm not speaking as someone with only theoretical knowledge here. I write and maintain high volume applications that make heavy use of multithreading, and I generally don't find the thread pool to be the correct answer.

Ah, argument from authority - but always be on the look out for people who might be on the Windows kernel team.

Neither of us were arguing with the fact that if you have some specific requirements then the .NET ThreadPool might not be the right thing. What we're objecting to is the trivialisation of the costs to the machine of creating a thread.

The significant expense of creating a thread at the raison d'etre for the ThreadPool in the first place. I don't want my machines to be filled with code written by people who have been misinformed about the expense of creating a thread, and don't, for example, know that it causes a method to be called in every single DLL which is attached to the process (some of which will be created by 3rd parties), and which may well hot-up a load of code which need not be in RAM at all and almost certainly didn't need to be in L1.

The shape of the memory hierarchy in a modern machine means that 'distracting' a CPU is about the worst thing you can possibly do, and everybody who cares about their craft should work hard to avoid it.

Share Improve this answer

Follow

answered Aug 13, 2008 at 22:08

Will Dean

39.5k ● 11 ● 92 ● 118

1

When you're going to perform an operation that is going to take a long time, or perhaps a continuous background thread. I guess you could always push the amount of threads available in the pool up but there would be little point in incurring the management costs of a thread that is never going to be given back to the pool.

Share Improve this answer

Follow

answered Aug 13, 2008 at 19:27

Quibblesome

25.4k ● 10 ● 62 ● 104

1

Threadpool threads are appropriate for tasks that meet both of the following criteria:

1. The task will not have to spend any significant time waiting for something to happen
2. Anything that's waiting for the task to finish will likely be waiting for many tasks to finish, so its scheduling priority isn't apt to affect things much.

Using a threadpool thread instead of creating a new one will save a significant but bounded amount of time. If that time is significant compared with the time it will take to perform a task, a threadpool task is likely appropriate. The longer the time required to perform a task, however,

the smaller the benefit of using the threadpool and the greater the likelihood of the task impeding threadpool efficiency.

Share Improve this answer

Follow

answered Apr 30, 2013 at 20:57

supercat

80.8k ● 9 ● 174 ● 220

0

MSDN has a list some reasons here:

http://msdn.microsoft.com/en-us/library/0ka9477y.aspx

> There are several scenarios in which it is appropriate to create and manage your own threads instead of using thread pool threads:
>
> - You require a foreground thread.
> - You require a thread to have a particular priority.
> - You have tasks that cause the thread to block for long periods of time. The thread pool has a maximum number of threads, so a large number of blocked thread pool threads might prevent tasks from starting.
> - You need to place threads into a single-threaded apartment. All ThreadPool threads are in the multithreaded apartment.
> - You need to have a stable identity associated with the thread, or to dedicate a

> thread to a task.

Share Improve this answer Follow

answered Aug 13, 2012 at 4:45

hwiechers
15k ● 8 ● 57 ● 62

-1

@Eric

> @Derek, I don't exactly agree with the scenario you use as an example. If you don't know exactly what's running on your machine and exactly how many total threads, handles, CPU time, RAM, etc, that your app will use under a certain amount of load, you are in trouble.

Are you the only target customer for the programs you write? If not, you can't be certain about most of that. You generally have no idea when you write a program whether it will execute effectively solo, or if it will run on a webserver being hammered by a DDOS attack. You can't know how much CPU time you are going to have.

Assuming your program's behavior changes based on input, it's rare to even know exactly how much memory or CPU time your program will consume. Sure, you should have a pretty good idea about how your program is going to behave, but most programs are never analyzed to determine exactly how much memory, how many handles, etc. will be used, because a full analysis is

expensive. If you aren't writing real-time software, the payoff isn't worth the effort.

In general, claiming to know exactly how your program will behave is far-fetched, and claiming to know everything about the machine approaches ludicrous.

> And to be honest, if you don't know exactly what method you should use: manual threads, thread pool, delegates, and how to implement it to do just what your application needs, you are in trouble.

I don't fully disagree, but I don't really see how that's relevant. This site is here specifically because programmers don't always have all the answers.

> If your application is complex enough to require throttling the number of threads that you use, aren't you almost always going to want more control than what the framework gives you?

No. If I need a thread pool, I will use the one that's provided, unless and until I find that it is not sufficient. I will not simply assume that the provided thread pool is insufficient for my needs without confirming that to be the case.

> I'm not speaking as someone with only theoretical knowledge here. I write and maintain

> high volume applications that make heavy use of multithreading, and I generally don't find the thread pool to be the correct answer.

Most of my professional experience has been with multithreading and multiprocessing programs. I have often needed to roll my own solution as well. That doesn't mean that the thread pool isn't useful, or appropriate in many cases. The thread pool is built to handle worker threads. In cases where multiple worker threads are appropriate, the provided thread pool should should generally be the first approach.

edited Aug 13, 2008 at 22:21

answered Aug 13, 2008 at 22:05

Derek Park
46.8k ● 16 ● 59 ● 76