

Re: Best way to design multithreading application

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework/2007-09/msg00385.html>

- *From:* Peter Duniho <NpOeStPeAdM@xxxxxxxxxxxxxxxxxxxx>
 - *Date:* Thu, 27 Sep 2007 11:55:07 -0700
-

Dave wrote:

Ok, I appreciate any help you can give me. I am somewhat new to threading, only understanding it conceptually, and I use the Threading namespace for the sleep() command. So, any suggestions with code would also be beneficial.

Basically, I need help designing how my threads should interact. Currently, the application acquires a bit of data, processes it, writes it to a file, sounds alarms if necessary, adds it to the UI (charts and tables), and then loops to acquire more. This results in data acquisition times of several seconds, and we need it under a second. The data acquisition is more important than UI updates, but not at the expense of UI updates never taking place.

The first thing to understand is that introducing threading may or may not improve performance. Usually it doesn't. If your code currently takes several seconds to acquire data, it will still take several seconds to do so even if you introduce threading.

The exception is for algorithms that are CPU bound and which can be parallelized, and even then only when you have multiple CPUs in the computer. In that case, you can have more than one thread working on the problem simultaneously, which can get the job done faster.

That said, performance is not the only reason to use threading. A classic example is to allow the user-interface thread to be responsive while some other processing takes places. This appears to be applicable to your situation, and there are implicit and explicit ways to use threading to address that situation.

Based on the information I know, here is what I would do (but don't know how): Thread A, which is high priority, sits there and acquires data over and over, adding it to a Queue 1 as it acquires more. Thread B, which is medium-to-high priority, loops over and over, and when it sees data in Queue 1, it writes it to a file and sets any alarm states. It then puts the data in Queue 2. Thread C, which is lower priority, loops over and over, and uses any data in Queue 2 to update the UI.

Re: Best way to design multithreading application

Is that the correct way to design a good solution for this? If so, I am worried about a few things:

1) If data collection is high priority, and the other threads cannot get enough CPU time, will either Queue fill up too quickly, effectively stalling the application (i.e. how would I "slow down" the acquisition thread if I notice the queues filling up)?

I would not mess with thread priority, at least not as an initial solution. When you have some task that is independent of other tasks, and which truly is lower or higher priority, then adjusting your priority can make sense. But when the threads involved are dependent on each other, as they are here, it makes little sense to make one thread higher priority than another, and for the reason you note: you really do need each of these threads to, at least on average, process the data at the same rate, otherwise you will eventually consume all your resources (memory in this case, though having the UI lag behind the actual processing is probably not desirable either).

2) Because of these extra queues filling up, am I actually taking more time to process each acquisition given the memory needs of such a system (i.e. should I keep it a single-threaded model)?

You should not create a design that involves queues "filling up". It's okay for there to be temporary bursts, creating some "peak" memory usage, but on average all of the threads need to stay in reasonable step with each other. Otherwise, the end result is consuming all of the memory.

So as long as you manage those bursts to minimize memory usage, there shouldn't be problem. It seems likely to me that you won't need to do anything special per se to "manage" the bursts. On a computer with a decent amount of RAM, you should have plenty to be able to deal with whatever buffering the i/o requires.

It seems to me that given your stated problem, some kind of threading is going to be desirable. There seem to be two different things you want to address:

- 1) You want to have a responsive UI even while some longer i/o operation is taking place
- 2) You want to have a thread that focuses only on data acquisition and which doesn't waste time saving the acquired data to a file

The three-thread solution you propose should address those needs nicely. There are other ways to address the issue, but I don't think that they are necessarily better or worse. Just different.

3) Is there even a way to prioritize threads in such a way that Thread A is always exactly (or rather, close to) 1 second per acquisition run, and then Threads B and C use up the extra time, with B being most important?

No. But if it makes sense for thread A to only do some i/o operation once a second, you can use the `Thread.Sleep()` method for that purpose.

Re: Best way to design multithreading application

4) What happens when Thread A enqueues a new chunk of data at the same time Thread B is dequeuing one? Same for B and C. Is this where the "lock" keyword comes into play?

Yes. For sure, if you're going to use threads, you need to synchronize them at some point, so that they aren't accessible the same data structures simultaneously. What you're talking about is often called the "producer/consumer" pattern. Any code that wants to use the queue needs to prevent other code from accessing the queue while it's using it. A handy class for this pattern is the Monitor class, as not only does it provide the enter/exit semantics the lock() statement does, it includes a wait/signal mechanism as well.

Jon Skeet has a nice chapter on threading here:

<http://www.yoda.arachsys.com/csharp/threads/>

I thought he had something on his web site that talked about the producer/consumer pattern, but I can't find it at the moment. Maybe it's buried in the above section, or maybe I'm just misremembering.

I also thought I had posted an example of a producer/consumer test, over in the m.p.dotnet.languages.csharp newsgroup, but I can't find that either.

Obviously I am going senile. If I come across it, I'll follow-up with a link. Otherwise, hopefully the above is enough to get you going in the right direction. :)

Pete

.