

Re: GetStream.Read behavior changed in .Net 2.0 with respect to ReceiveTimeout

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework/2007-08/msg00014.html>

- *From:* Keith Langer <tanalbit@xxxxxxx>
 - *Date:* Wed, 01 Aug 2007 12:52:24 -0000
-

On Jul 31, 11:56 pm, Peter Duniho <NpOeStPe...@xxxxxxxxxxxxxxxxxxxx> wrote:

Keith Langer wrote:

Could you show me some code which uses these alternatives for checking if data has arrived?

For the record, I only mentioned alternatives to implementing a timeout. Alternatives for "checking if data has arrived" is a somewhat different topic.

The only other way I know to check for data on the socket is to keep checking the DataAvailable property and sleeping for a small period of time, then exit out when my timeout period has elapsed without receiving data.

I guess at this point it may be helpful to take a step back and look at the bigger picture. From the sounds of it, you are using a timeout for something other than an actual timeout. It might be helpful to be more clear about why it is you are using a timeout in the first place. Most socket i/o code does not bother with a timeout at all.

For simple socket-based applications, in .NET or otherwise, generally the design either assigns a dedicated thread to each socket for receiving, or uses the function/method named "select" or "Select" (Winsock and .NET Socket, respectively). The thread-per-socket model works fine for very small numbers of sockets, while the select model works fine for slightly larger numbers of sockets, but less than 64 (in Winsock, there's a limit of 64 sockets that can be passed to select...I'm not sure if the same limit exists in the .NET version, but it may).

In .NET specifically, one would normally use the async paradigm. That

Re: GetStream.Read behavior changed in .Net 2.0 with respect to ReceiveTimeout

is, using the BeginXXX/EndXXX methods for i/o. These are the most efficient, and for very large numbers of sockets they are the only practical way to produce scalable network i/o code. (In Winsock, there are other alternatives to deal with larger numbers of sockets, one of them being to use i/o completion ports, which is what the .NET async paradigm is built on when run on platforms that support it).

IMHO, even if you have just one socket, the async paradigm makes a lot of sense. It is easily extended if you wind up needing to do so in the future, it has no additional issues beyond that which would normally be present in a dedicated-thread implementation, and personally I just like the way it works. Via a BeginXXX method you tell .NET to call a specific method when the i/o completes, and when it completes, your method is called. You do in there whatever you want to process the i/o (such as calling Socket.EndReceive() for a receive operation), and then you call the BeginXXX method again to repeat the sequence.

In any of these mechanisms, there is no need for a timeout. You don't do your i/o on a thread that needs to do something else. Using Socket.Select(), you can in fact provide a timeout value, but that shouldn't be used just so that you can mix your i/o code in the same thread with code that does something else.

Anyway, that's all a long way of saying that, from the limited amount of information you've posted so far, it sounds to me as though even though you are asking how to get the timeout to do what you want, the more basic issue is that using a timeout isn't the appropriate solution in the first place.

So, if after reading the above you still think it is, maybe the next step here is for you to explain in more detail why you need the timeout. That would help with respect to answering the question regarding how to implement a timeout.

If instead the question remains "Could you show me some code which uses these alternatives for checking if data has arrived", then hopefully the above at least begins to answer that question (the code is on MSDN, as sample code for the various methods and other members of the Socket class).

Finally, yes...I realize you asked about GetStream.Read() (which is a little confusing itself...I know you really mean Stream.Read(), but since there's not actually any GetStream class, saying "GetStream.Read()" seems a little odd out of context), but clearly what's really going on is related to the socket itself, not the stream wrapped around it, I think it's more useful to discuss the socket specifically. You can implement the above either by manipulating the socket directly, or in the case of the BeginXXX/EndXXX method, the Stream class does have the async i/o methods in it, so you can operate on the stream directly in that case.

Pete

Pete,

This application runs a number of signs (serial devices) connected to several device servers. Each device server has a dedicated thread and TcpClient associated with it in my application. There could be anywhere from 1 to 150 of these device servers, and each device server could have 1 to 100 signs (a typical site might have anywhere from 10 to 500 signs in total). Some of the signs respond to messages sent, but there is no guarantee that all signs are functional. So for each device server, I must send a message to sign 1, wait for a response or for a timeout period of X seconds to elapse, then send the next message to that sign or the next sign. Hence, this is my need for some sort of timeout. I either need to keep checking the DataAvailable property for several seconds, or implement a blocking operation for several seconds until data is available.

Keith

.