

Re: OutOfMemoryException from Thread.Start

Source:

<http://www.tech-archive.net/Archive/WindowsCE/microsoft.public.windowsce.app.development/2005-07/msg00079>

- *From:* "Paul G. Tobey [eMVP]" <ptobey no spam AT no instrument no spam DOT com>
 - *Date:* Thu, 14 Jul 2005 08:28:15 -0700
-

You don't have to implement your own resend process if you're using stream/TCP sockets. That's all handled by the TCP layer of the network stack. In fact, by doing it yourself, you're quite likely to screw things up and get multiple copies of things at the other end. Also, your algorithm does **not** match TCP standards.

As for the out-of-memory, I'm not sure what might be going on, but you should be sure that any previous threads that have been fired and are done properly exit. You might be running out of thread handles or something like that, although that seems unlikely.

You might try asking the out-of-memory question in microsoft.public.dotnet.framework.compactframework, as that's where most of the managed code experts are located.

Paul T.

"Michael--J" <MichaelJ@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message news:D729402B-124B-434E-9325-9D5AE0E26F1F@xxxxxxxxxxxxxxxxxxxx

> Hi,

>

> I have an embedded board with a PXA 270 chip. It has a Win CE 5.0 OS image

> loaded and it supports CF v1. My task was to write a component which

> applications will use to transmit objects/structures to other application

> across the network. This component will be used by applications running on

> the board i mentioned above as well as in normal PC applications. This is

> the

> reason why i couldn't use .NET Remoting - CF does not support it.

> Consequently, i have written a component in C# which provides the

> following

> functions:

> - send a structure to a remote endpoint

> - receive a structure from a remote endpoint

>

> This is how i have implemented it:

> - On sending, the component serializes the object in wants to send into a

> stream of bytes (packet) first and then sends it by calling the Socket

Re: OutOfMemoryException from Thread.Start

- > object's synchronous Send() method. Before returning, this send process
- > needs
- > to make sure that the other end received the packet (this is where the
- > reliability part comes into the picture). To know that a packet was
- > received
- > correctly, the send process waits 200ms (RTO – ReTransmissionTimeout) for
- > an
- > ACK packet to come back (the reception of this ACK packet is explained
- > below
- > in the receive section**). If it doesn't receive an ACK by then, it
- > resends
- > the same packet. It continues to resend the packet every 200ms for 30
- > secs.
- > If it still hasn't received it by then, it disconnects the connection to
- > the
- > remote endpoint.
- >
- > – On receiving, bytes are received asynchronously by calling the Socket
- > object's BeginReceive() and EndReceive() methods. The receiving process
- > blocks until bytes become available on the receiving port. So when bytes
- > do
- > become available on the receiving port, they are read and deserialized
- > back
- > to the original object. In order not to miss any further incoming bytes,
- > the
- > currently received object is passed to another thread for processing.
- > Hence
- > the receiving process can return and call BeginReceive() again and wait
- > for
- > more bytes to come in.
- > Now, the new thread that just got spawned will process the packet.
- > This
- > component is now being tested with a TestServerApplication (runs on my dev
- > PC) and a
- > TestClientApplication (runs on the PXA board). The server simply waits for
- > clients to connect. When
- > the client connects, the server accepts it and waits. The system was set
- > up
- > such that the client app sends a 2KB packet to the server and the server
- > basically echoes it back. When the client receives the echo, it also
- > echoes
- > that back and so they play "packet tennis". This is simply to test the
- > component. The size of an ACK packet is 44 bytes. So this is what happens:
- >
- > – client sends 2KB to server
- > – server receives and sends a 44-byte ACK, and then the echo 2KB packet to
- > the client
- > – client receives the ACK (which ends the first send), and then also
- > receives the echoed packet... client then sends an ACK to the server, as
- > well
- > as the echo packet

Re: OutOfMemoryException from Thread.Start

> – and the process repeats...
>
> Now, the problem i am having occurs on the PXA side. At some random point
> in
> time, when a structure is received and it is passed to another thread for
> processing, an OutOfMemoryException occurs when calling Thread.Start(). So
> i
> thought, ok maybe there isn't much RAM, but then i ran it again with the
> RAM
> settings opened (the Control Panel item showing how much Storage Memory
> and
> Program Memory the system has). While the app runs, the amount of RAM
> decreases but when the exception occurs, there seems to be a few MBs left
> of
> program memory... how could this exception have been raised? Thanks...
>
> I have written a client application in C# The board will be used to run a
> client application which will connect

- **Follow-Ups:**

- ◆ **[Re: OutOfMemoryException from Thread.Start](#)**

- ◆ *From: Michael--J*

- **References:**

- ◆ **[OutOfMemoryException from Thread.Start](#)**

- ◆ *From: Michael--J*

- Prev by Date: **[Re: using TAPI](#)**
- Next by Date: **[Re: IPC mechanisms supported in Windows CE](#)**
- Previous by thread: **[OutOfMemoryException from Thread.Start](#)**
- Next by thread: **[Re: OutOfMemoryException from Thread.Start](#)**
- Index(es):
 - ◆ **[Date](#)**
 - ◆ **[Thread](#)**