

Re: Socket weirdness

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework/2006-09/msg00477.html>

- *From:* "Alan J. McFarlane" <alanjmc@xxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Tue, 19 Sep 2006 12:15:14 +0100
-

In article news:O1OxDg52GHA.1588@xxxxxxxxxxxxxxxxxxxxxx, William Stacey [MVP] wrote:

"Dave Sexton" <[dave@jwa\[remove.this\]online.com](mailto:dave@jwa[remove.this]online.com)> wrote in message news:O6rzGL42GHA.5048@xxxxxxxxxxxxxxxxxxxxxx

[...]

The other thing we discover here via imperative evidence is that shutdown.receive will never be sent by server unless it can piggyback on an ACK reply or an outgoing message. Something I must remember. Cheers.

I haven't tried using an outgoing message. Did you test that?

I guess the bit is not set in a send either – only the ACK. So I guess that means you must always do at least two sends (at the client) before you will notice a shutdown receive at server.

What "bit"? There is **no** flag for Receive_Closed! There are two relevant flags: the FIN flag which means Finished_Sending, and RST, which means Reset: in general means 'I've no such connection, please delete that connection from your list'. The flags SYN, ACK, PSH, and URG have no part to play here.

So when the server does "socket.Shutdown(SocketShutdown.Receive);", it can't do anything based on that. It just sits dumb waiting for another action. Then eventually a packet comes from the peer, and that will contain data, so the server responds RST: 'I can't handle that'. This is I presume what Dave meant when he said "you are basically setting up a wall": no signal, just that any data received will crash into the wall. However he wasn't quite correct in saying 'It would be nice, however, if the first send failed as well but I see that from the docs this "design" was by choice.' The software writers have *_no_* choice in this, the TCP protocol doesn't allow such a behaviour.

So, remembering that the server has sent nothing after the local application did SocketShutdown.Receive. Lets

Re: Socket weirdness

look now at what happens from the aspect of the client side. Remember that TCP makes no guarantee of maintaining send boundaries etc. So the client could buffer up multiple send before deciding to send a packet, that packet then works its way across the network, meanwhile the client application could be doing more sends, which are likely buffered up, and maybe sent at some point, and again the client application could be doing more sends, and even blocking or non-blocking :-,) on a receive.

And at some point the first send now reaches the server, who responds with a packet with the RST flag set. That then makes its way back across the network. Again the client application can be doing sends, and maybe some packets are sent too. Then the RST reaches the client device, and works its way up through the stack. Boom! Now and only now is ConnectionReset returned on every subsequent send or receive that the client application makes.

So two things about TCP, firstly SocketShutdown.Receive cannot be signalled. That's a very rare and odd case that there's no need to cover it: 'I've more to send, but I don't want to hear any error responses from you!' Odd, whereas the opposite is common: 'I've now finished sending all I have, but I'm still listening for any feedback you wish to send me.'

Secondly, that *errors* on the connection can only be signalled on later send/receive calls. Errors are only apparent after buffered data has been sent and a error response has been received from the server across the network, or if the network is down for instance that 'I've had to retry sending data five (say) times and I've now given up'.

Note that SocketShutdown.Send, is signalled with flag FIN, which is 'data' so it is sent in a packet with or after the last byte in the send buffer has been sent. To get an error earlier some protocols send regular packets to say 'are you alive?' (for instance HDLC, IrDA, SNA, SPX (?my memory is fading)). But due to the philosophy of TCP, that, to try and ride over network failure do not send any packets unless there's data to be sent, no such packets are sent.

I recommend that you put your server and client on different machines and look at the traffic between them with a network analyser. What's happening below the APIs can then be seen.

—

Alan J. McFarlane

<http://www.alanjmcf.me.uk/>

Please follow-up in the newsgroup for the benefit of all.

.