Subject: Re: Checking socket connection after send.
Posted by rylek on Wed, 20 Feb 2008 19:59:26 GMT
View Forum Message <> Reply to Message

Hello captainc!

I'm afraid the answer to your question is not altogether simple. Although it might sound arbitrary and inexact, perhaps the most important auxiliary question is: "how certain do you want/need to be that your request has been processed on the remote computer?"

That, of course, depends on the character of your application. In many situations it is quite sufficient to know that the data has probably left your computer. Even this is not really simple; Windows has large outgoing data buffers (although I think their size can be changed using some setsockopt or perhaps ioctlsocket) so that you can place a lot of data in an output socket without getting a write block even though the connection will break shortly and none of the data will eventually leave the machine.

Now when you're writing a transaction-sensitive tool, like a phone-based credit card terminal system, that's something completely different. There are a million evil things that can befall a solitary packet floating around the cyberspace after being spat out of your machine's network card. It can get lost on its way, it can get rejected by a server application that's just crashed, there can be a power failure while the remote machine processes the packet etc. etc. If you don't want spend the rest of your life paying losses to the bank using your system, you had better wait for an acknowledgement response.

This is the sad truth of remote computation: while both machines are talking, you know everything's fine. One machine stops talking, the other has no way (without restoring the communication) to find out what part of communication since the last handshake has got through the line and through the remote application.

There's a similar problem with disconnection: you can never be absolutely sure that the other side has disconnected gracefully, because each side would have to wait for the other side's confirmation and both machines would deadlock forever. The basic rule is that enough bidirectional communication must have taken place to make sure on both sides that the potentially ungraceful shutdown will not have serious negative effect on either side.

A somewhat stricter formulation: the communication protocol and its implementation should always ensure that an ungraceful remote shutdown at any time doesn't endanger stability and transaction data on either side.

Regards

Tomas

P.S. As concerns PeekAbort, this is supposed to check whether the remote machine has closed the socket. The implementation technique is not very clean (if you know of a cleaner way, I'm prepared to listen), it just tries to receive a single byte on the line. If the line has broken, the recv function returns 0. However, when the connection stays alive, the recv reads the one byte which

has to be stored in the leftover data buffer, so that a repeating call to PeekAbort without actually receiving anything (while the remote application is sending data) will cause unlimited growth of the leftover data buffer possibly leading to application memory overflow.
