Subject: Checking socket connection after send.
Posted by captainc on Wed, 20 Feb 2008 15:51:38 GMT
View Forum Message <> Reply to Message

I need to find the best way to check if data was sent & received successfully without implementing a return message by the receiving side. My thought has been to write the message to the socket, see if I can still write to the socket, and if I can, I can know that the message was sent successfully without transmission interruption. If I cant still write, then I know that the connection has closed prematurely. Also, I will check for errors, but the IsError() function has not detected dropped connections for me.
This would look something like:

```
socket.Write(message);
if(socket.PeekWrite()){
  socket.Close();
  return true; //
}else{
  socket.Close();
  return false;
}
```

Is this correct? Wouldnt the Write() method cause a Socket Error to be set if it could not write the whole message?
Am I interpreting PeekWrite correctly? Does it check to see if the socket is writable or if there is content waiting to be written?
Also, what does PeekAbort() do?

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.

Subject: Re: Checking socket connection after send.
Posted by captainc on Wed, 20 Feb 2008 21:32:46 GMT
View Forum Message <> Reply to Message

What about the SO_KEEPALIVE option? I've been doing some reading on this. Could this be used in setsockopt() to detect connection breaks?

Mainly my issue is this... I am using socket to send files from one machine to another. I can disconnect the connection in the middle of the transmission whereby the sending side finishes sending without a problem, but the receiving side doesn't receive the full content and gets an error (IsError() --> 10054 socket(272) / recv: Connection reset by peer. (WSAECONNRESET)) The sender never has the chance to detect the error before finishing its operations, which in my case, it deleted the source file because it thinks it is done with it.

Subject: Re: Checking socket connection after send.
Posted by mr_ped on Thu, 21 Feb 2008 08:54:24 GMT
View Forum Message <> Reply to Message

As deleting data is involved, you can't do this without proper acknowledgment from receiver after checking the data are properly stored and doing probably some CRC/HASH check to be sure no bits got corrupted during network transfer.

Either that, or leave the check & deletion on the user. Or accept the chance of losing data (just like the MS home server does it).

Subject: Re: Checking socket connection after send.
Posted by captainc on Thu, 21 Feb 2008 13:33:07 GMT
View Forum Message <> Reply to Message

How much of the data checking is done by the protocol (TCP)? Doesn't it do packet-level crc'ing? I didn't think it necessary to implement the crc as I thought the network layer for connection-oriented transfers would handle that.