

Re: Interesting TCP behaviour with large sends/small buffers

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.networks/2007-10/msg00074>

- *From:* "Alexander Nickolov" <agnickolov@xxxxxxxx>
 - *Date:* Tue, 23 Oct 2007 17:43:29 -0700
-

I haven't seen this behavior before, and you can't make such conclusion. The implementation of select is naturally not documented. Of course it would use WaitForMultipleObjects, not WSAEventSelect, since the latter is called once per socket...

--

=====
Alexander Nickolov
Microsoft MVP [VC], MCSO
email: agnickolov@xxxxxxxx
MVP VC FAQ: <http://vcfaq.mvps.org>
=====

<barkan.alon@xxxxxxxx> wrote in message
news:1193179178.319116.171640@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

The program was originally intended for simulating a digital i/o to TCP converter I've been working on, in which data arrives at specific constant bitrates and not as fast as TCP can handle it. The multithreaded design is intended for keeping the data rate close to the required rate. Since I ran into the problem mentioned above while working on that program, I tried to investigate it using the same code base.

I'll try your workaround and let you know how it went.

BTW, have you ran into this bug before?
And is it correct to conclude from your explanation that 'select' is internally implemented using WSAEventSelect?

Thanks,
Alon

On Oct 23, 7:06 pm, "Alexander Nickolov" <agnicko...@xxxxxxxx> wrote:

Try sending without waiting on select and only wait when send returns with error and status of WSAEWOULDBLOCK.

Re: Interesting TCP behaviour with large sends/small buffers

This seems to be a bug in the select implementation rather than a TCP issue. By not waiting in the beginning of your cycle you should be able to work around it. Note what I'm proposing is the semantics for WSAEventSelect.

BTW, why don't you simply write as fast as you can and get rid of your complex logic with semaphores and such?

--

=====
Alexander Nickolov
Microsoft MVP [VC], MCSD
email: agnicko...@xxxxxxxxx
MVP VC FAQ:<http://vcfaq.mvps.org>
=====

<barkan.a...@xxxxxxxxxx> wrote in message

news:1193090408.595648.279010@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Hello,

I apologize for the long post, but I've recently encountered an interesting phenomenon:

I wrote 2 programs: One that acts as a TCP server, and the other as a client. The server, upon connection, sends a configurable number of bytes to the client every 20ms (on average). This is done by one thread that raises a semaphore every 20ms, and another thread which waits on the semaphore and then sends. The server uses a 'send loop' to send – it repeatedly calls 'select' for writability and then a nonblocking 'send' until the required number of bytes has been sent. The client program simply receives the data and once per second prints how much data was received. The idea was to see how close the transfer rate could get to wire speed (a little over 94mbps for TCP/IP over a 100mbps ethernet) and

Re: Interesting TCP behaviour with large sends/small buffers

how large the send buffer should be to achieve that.
I set the client's receive buffer size to 1MBps, to make sure it won't be a bottleneck.
Each program ran on a different machine, both connected to the same switch.
I then tried the server with various combinations of send buffer sizes and 'bytes per send loop'.

With a 1MBps send buffer, I reached 94Mbps without a problem (235KB per 'send loop').
I then tried with a send buffer of 200KBps, but the throughput was much lower this time – almost 10 times lower. This took me by surprise since the bandwidth*delay product is much lower than 200KBps, even if assuming an RTT of 10ms (which is quite high for a same-switch connection).

While trying to investigate this, I tried with 50KB per 'send loop' (20Mbps – 50KB every 20ms) and a send buffer of 50KB. This still produced a low throughput.
I then used WireShark on the client side, and found this interesting behaviour: After the sender sends the last TCP packet of the current 'send loop' (i.e. every 50KB), it waits for an ACK for that last packet before sending the next packet. However the client delays the ACK, according to the delayed ACK algorithm – 50KB bytes means 34 MSS-sized packets and another 360 bytes sized packet, which is an odd number of packets, which explains the delayed ACK. This 200ms delay explains the low throughput – instead of sending 50KB every 20ms, the server waits 200ms before it continues sending!

Re: Interesting TCP behaviour with large sends/small buffers

To make sure that the server really is waiting for the ACK, and that nothing else is causing the 200ms delay, I temporarily disabled delayed ACKs (somewhere in the registry), and the problem disappeared. I also tried sending exactly 34 MSS-sized packets each time, so that the ACK for the last packet won't delay, and the delay was gone.

I also checked this on the server side by 'printing' how much time the 'select' call waits for writability each time it is called. The results were as such: The first 'select' of each 'send loop' waited approximately 200ms (except for the first 'send loop'). Also, on the last packet of every send, the PSH bit is set (according to the WireShark dump). AFAIK this means that it's the last unsent packet left in the buffer. So considering that ACKs for all the previous packets have been sent (and probably received during the 200ms delay), surely there should be available buffer space in the send buffer and the application isn't supposed to wait 200ms. This means that not only TCP doesn't send any data until the ACK for the last packet arrives, but the application can't insert data to the send buffer during that time!

Investigating some more, I found that: When trying to send 50000 bytes each time with a send buffer of 50001, the sender doesn't wait for an ACK for the last packet. The same happens with any buffer that's larger than 50000 bytes. When trying with any buffer that's smaller than 50000, the server always waits for an ACK.

I since tried with various buffer sizes and send rates, with results

Re: Interesting TCP behaviour with large sends/small buffers

seemingly according to the following rule: Whenever the send size is larger than the send buffer, the sender waits for an ACK for the last packet of the current send before sending more data and before allowing more data to be inserted into the buffer. AFAIK This is not the way TCP is supposed to behave. This is a rather unpleasant problem if the send is composed of an odd number of packets

- the sender waits for a delayed ACK after each send.

Some more information which may be related:

- Running WireShark on the server, the packets seem to be sent in chunks much larger than the MSS – sometimes as much as 30 times the MSS. After researching a little I found this is probably due to the server using 'segmentation offload', i.e. the segmentation is done by the network adapter, which is common on Windows.
- Both the machines are running Windows XP SP2.

I'll be glad if anyone can shed some light on this issue. And again I apologize for the long post. :-)

Thanks,
Alon