

## Re: Sockets debugging tools

**Source:** <http://unix.derkeiler.com/Newsgroups/comp.unix.programmer/2005-02/0399.html>

---

**From:** T Koster (*reply-to-group\_at\_use.net*)

**Date:** 02/16/05

Date: Wed, 16 Feb 2005 14:23:55 GMT

Andrei Voropaev wrote:

> *On 2005-02-16, T Koster <reply-to-group@use.net> wrote:*

> *[...]*

>

>>*However, since my application uses non-blocking sockets and the select  
>>system call, I need to be able to test conditions like when send doesn't  
>>manage to send an entire message, or its buffer fills (through lack of  
>>or delayed ACKs from the peer due to poor network conditions) and  
>>reports that it would block. In these cases, my application \*should\* be  
>>storing the remaining portion of the message in its own user-space  
>>buffer to send later, when the select system call says that it's okay to  
>>send again. Currently I have no way to force this behaviour without  
>>coding some sort of command to the application protocol almost like  
>>"ping -f" in behaviour, to try to flood things.*

>

> *You are wrong. This is perfectly possible by programming your peer not  
> to read from the socket :) (Emulating the deadlock of the peer). In this  
> case after the system buffers are filled, your call to send will process  
> less (or even none) of the data. Note, system buffers are fairly large  
> so you may need to pass lots of data before they fill up. But they do  
> fill up. Tested :)*

Hmmm okay. So to start filling up the system's I/O buffers I just freeze the client program, and watch how my program handles send's inability to send?

If so, what would happen if the client program never springs back to life, and the socket just remains in a permanent would-block state? After a while the OS will time-out the connection, right? Where is this error raised? The tcp(7) man page only specifies ETIMEDOUT but no indication of which calls raise it. Is it send and friends?

>>*So, there is a possibility that my program is too paranoid about failure  
>>to send a message by the OS? It is true that almost all sample sockets  
>>programming code I find on the 'net doesn't seem to worry too much about  
>>it. I've been trying to read through some IRC server source code, to  
>>see how they do it, considering they generally handle giant loads, but  
>>so far they all seem to stem from the same ancient ircd and the code is*

>>fairly messy and difficult to read. I'll get there eventually. Luckily  
>>this is a hobby project for me or I would have exceeded a hundred  
>>deadlines already :)  
>  
> Well. Your program should be ready to handle any error conditions  
> returned by `recv` or `send` calls. (If you use `select`, `poll` or similar,  
> then you have to watch for error flags there as well). See `man 7 tcp`,  
> `man 7 ip` for the errors. Since TCP is reliable protocol, the only thing  
> that may prevent it from sending data is an absence of the route to  
> peer. You can emulate this either by pulling the network cable out, or  
> by changing routing tables (not tested :) Anything else in the network  
> should not be a concern for you. (Though others may prove me wrong :)

In addition to the ETIMEDOUT error, I'm confused about whether I need to look out for EPIPE or not. One man page (`send(2)`) tells me it is raised when "the local end has been shut down on a connected socket." Surely the local end only shuts down if I call `close` or `shutdown` on the socket, right? Thus, I already know when the local end has been shut down and such a socket would never be sent to anyway, or am I missing something?

Another man page (`tcp(7)`) has something else to say about EPIPE: "The other end closed the socket unexpectedly or a read is executed on a shut down socket." This is totally different from what `send(2)` has to say, but sounds more likely. `ip(7)`'s story is similar to `tcp(7)`, but not identical, plus I find the following unreassuring note under the BUGS section: "There are too many inconsistent error values." :(

ECONNRESET is elusive too: neither the `ip(7)` or `tcp(7)` man pages mention it. It only appears to be mentioned in `send(2)`. Can it be expected to be raised by `recv` also? It makes more sense to me that `recv` should find out that a connection has been reset by the peer than `send`.

I would never have thought sockets programming on Linux would be so poorly-documented. I'm starting to read some FreeBSD man pages to see if they contain the details I'm missing.

Thomas