

Rust 'serialport-rs' Transmit-Receive Timing and Functional Test Utility

Serial port basic test application 'receive_timing_info'

This document describes the newly generated timing characterization test application for the **serialport-rs** crate. The test application is named **receive_timing_info**, and was created in support of Github issue #106. Using this new test application has assisted me in developing and testing a simple patch for the crate's Windows **read()** method, which corrects an issue under Windows where with a zero timeout setting if the read doesn't receive all requested data (per the supplied buffer's length), it blocks indefinitely. Hopefully this patch can be integrated into the next serialport-rs crate release.

The test application's source is contained in a single file named '**receive_timing_info.rs**', which one can easily build using cargo by copying it into the serialport-rs project's '**examples**' sub-folder. It has been tested on both Windows 10 and Ubuntu 22.04 LTS (both x64 Intel hardware). In theory it should run on any platform with standard support for the standard rust compiler, its build tools, serialport-rs its dependent crates, and three additional crates.io crates which '**receive_timing_info**' utilizes – described below. *If any of these current three dependent crates should be unavailable on another desired target platform (not the case with Ubuntu Linux and Windows), then it will be straight-forward to conditionalize those few lines of code in 'receive_timing_info.rs' which reference these additional dependent crates so the new target platform builds conditionally using alternate crates and/or alternate equivalent code sequences. This can be discussed as needed.*

To build 'receive_timing_info', follow these steps:

- a. Using 'git', clone the crate project source for **serialport-rs** from the Github repo at <https://github.com/serialport/serialport-rs>.
- b. Copy the new '**receive_timing_info.rs**' source file into the cloned crate project's '**examples**' sub-folder.
- c. This new test application currently requires a few additional dependencies beyond those which **serialport-rs** crate presently requires. The additional dependent crates are '**log**', '**fast-log**', and '**spin-sleep**'. One may add these additional dependencies to **serialport-rs** crates's '**Cargo.toml**' file by submitting the following three commands from the crate project's top-most folder while in a terminal command window =>
 - i. '**cargo add log**'
 - ii. '**cargo add fast_log**'
 - iii. '**cargo add spin_sleep**'
- d. Then build **receive_timing_info** test application by submitting the following commands, also while in a terminal command window with your working directory set to the top-most folder of the project.

'cargo clean'

'cargo build --example receive_timing_info' or **'cargo build --release --example receive_timing_info'**

- e. The resulting executable file will be located in the project's '**target\debug\examples**' or '**target\release\examples**' sub-folder respectively.

Next is an example MS-Windows terminal window command line that launches the new test application =>

```
>receive_timing_info.exe --txport=COM5 --rxport=COM6 --baud=115200 --log=D:\filename.log --rxtmo=20 --posttxdelays=0 -xfrstalledtmo=12000
--txlen=10 --rxlen=20 --repeat=10 --fulldbgy
```

The following table shows the available command line arguments that the 'receive_timing_info' utility accepts as command line arguments at launch. Note that in this test version the serial flow control is automatically set (by default) to None, and therefore the test doesn't test any OS level flow control functionality.

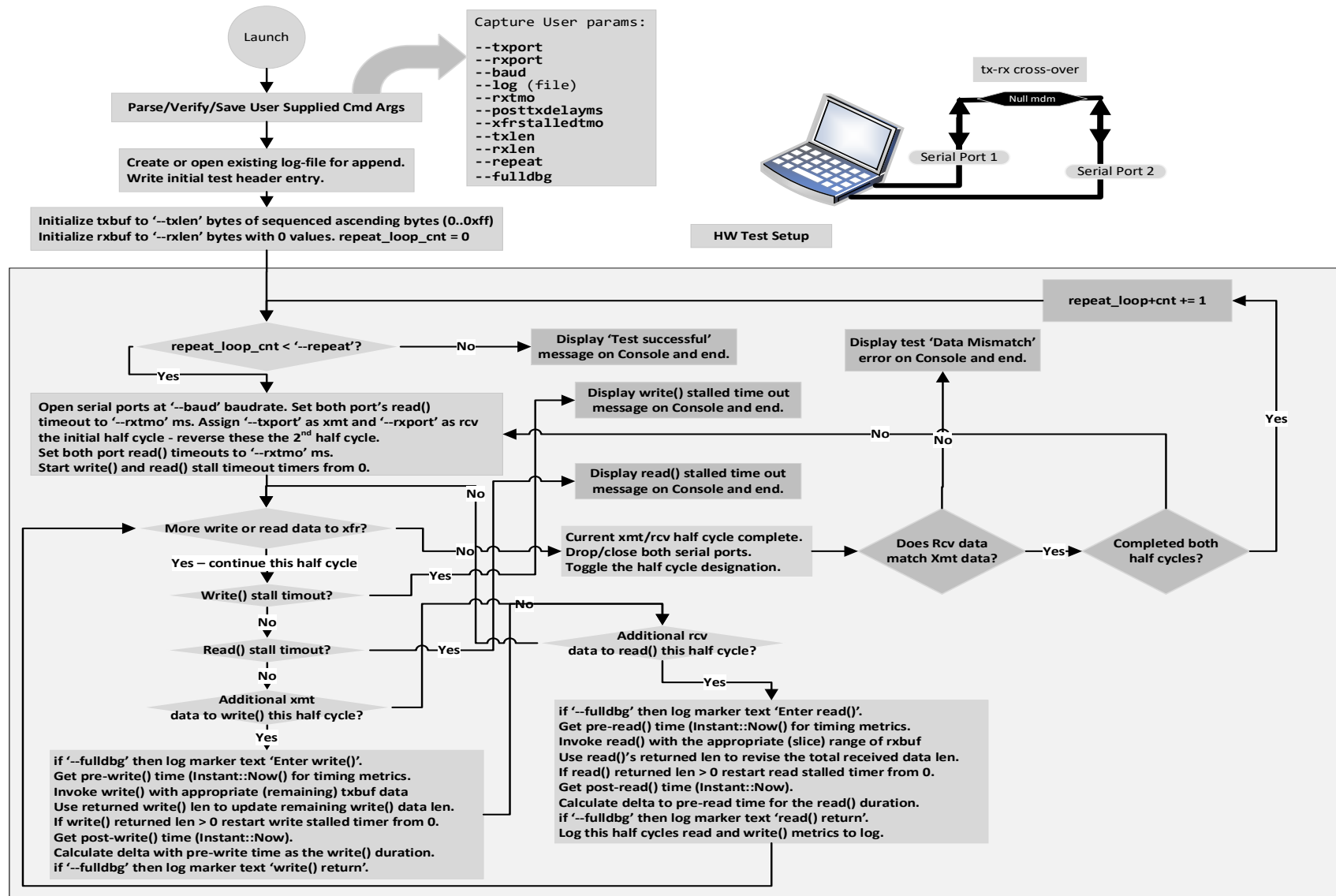
Syntax	Switch Purpose	Required or Optional
--txport=port-name	Platform specific port name (i.e. <i>COM4</i>)	Required
--rxport=port-name	Platform specific port name (i.e. <i>COM5</i>)	Required
--log=file-path	Platform specific log-file path	Required
--baud=bbbb	Baud rate (integer) – typical values are <i>9600, 115200, 200000</i> etc...	Required
--rxtmo=tttt	read() timeout in ms (integer)	Required
--posttxdelays=tttt	The delay in ms to wait after write() before invoking the corresponding read()	Optional (default=0)
--xfrstalledtmo=tttt	The time-period before aborting when an in-progress test transfer sequence of consecutive write() or read() invocations time-out repeatedly, when additional transfer data is expected. Separate timers are internally maintained for write() and read() . The respective transfer stalled timer is reset when any write() or read() in the sequence returns success and a positive 'len' value. This parameter's used value will automatically be set to the greater of 10000 (the default when not specified), or ' rxtmo x 4' when ' rxtmo x 4' is greater than both the default (10000) and the specified value. Note that the value actually used is indicated in the launch-time initial displayed informational text line.	
--txlen=nnnn	Specifies the total number of bytes to write() each half cycle. This byte sequence is an auto-generated [u8] array buffer with an ascending order byte pattern sequencing from 0 .. 0xff , repeating until exactly ' txlen ' bytes are generated. This buffer is then supplied to the write() invocation.	Required
--rxlen=nnnn	Specifies the number of bytes requested when initially invoking read() following its associated write() operation in a given half cycle. While outstanding transmitted data has not yet been received, read() will be invoked repeated until the lesser of ' rxlen ' and ' txlen ' total bytes of accumulated data have been read, or until an ' --xfrstalledtmo ' timeout occurs. Subsequent read() invocations (following the initial read() in the same half-cycle use	Required

	<p>a request length that is reduced in magnitude downward from 'rxlen' by the count of bytes already received.</p> <p>'txlen' and 'rxlen' are commonly specified with the same positive value when primarily testing for transfer throughput, data transfer integrity and timing metrics. Note that these two parameters may be specified with different values, which has benefits for certain testing purposes. To test the read() timeout behavior and its platform specific timing characteristics, specifying values for which 'rxlen' > 'txlen' will (or should!) result in read() timeout errors, before an eventual 'xfrstalledtmo' timeout eventually halts the test. One caveat for the current Window's crate is that this test scenario blocks the test application indefinitely when the read timeout parameter is set to 0 ('--rxtmo=0'). Specifying with 'rxlen' < 'txlen' parameter values should return 'rxlen' bytes successfully, while leaving 'txlen' – 'rxlen' bytes unread in the read serial port's input buffer (which is not a problem).</p> <p>Finally note that specifying a 'txlen' value larger than ~7500 on Windows will likely result in lost or corrupted transfer data. This is due to data buffer overruns in the platform OS layer, and is a by-product of the current test application's simple implementation/design. Similar data overruns may also happen on Linux, but at a larger 'txlen' value: I haven't yet characterized this under Linux. To elaborate, currently the test's blocking write() invocation executes prior to the initial invocation of the blocking read(), with both occurring inline in the test application's main() thread. This current simple design does not support separately threaded and overlapped write() and read() invocations. This can be added in a subsequent version, or possibly in a separate new test program.</p>	
--repeat=nnnn	Total number of read/write full cycles. Note that each full cycle consists of two (2) half-cycles, where each half-cycle writes ' txlen ' bytes to one of the two serial ports and (attempts to) read ' rxlen ' bytes from the other, while the second half-cycle reverses the direction of transmission with the two serial ports. Note that both ports are closed and reopened at each half-cycle transition. The closing and reopening of the ports at each cycle (or half cycle) may be eliminated or altered in a future release, or in a new test app.	Optional (default=1)
--fulldb=Y (or N)	<p>'Y' enables additional debug log messages to be generated during the test. This flag should normally be 'N' (disabled), since enabling it results in undesirably lengthening certain timing metrics. 'N' is its default setting.</p> <p>Enabling the flag is useful when a specific test execution hangs indefinitely (which is abnormal), and with it enabled the run-time test logic inserts additional log-file 'entering' and 'returning' log marker text lines before and after each run-time invocation of 'write()' and 'read()'. Then one can examine the resulting log-file after the hang up occurs, and its</p>	Optional (default=N)

	<p>clear which specific invocation is the cause of the hang-up. This is mainly useful in theorizing whether the hangup is due to faulty hardware, a bug in target specific crate code, electrical or cabling issues, or a previously undiscovered bug in the test software.</p> <p><i>Note that 'read()' or 'write()' blocking indefinitely could be detected with a separate monitoring thread, and an error message displayed for the test operator at execution time. This requires re-implementing the test with a multithreaded design (or in a separate new test application).</i></p>	
--	--	--

<This space left intentionally blank>

'receive_timing_info' Test Logic Flow-Chart



Description of v4.2.1 serialport-rs behavior for 'read()' and 'set_timeout()' methods with Linux and MS-Windows platforms

I could not find online documentation for 'serialport-rs' that explicitly describes the crate's **read()** method's expected timeout behavior for various scenarios related to a paired **read()** and **write()** per the supplied buffer parameter relative sizes and the **read()** end-point's timeout setting. Therefore, I ran experiments after examining the crate's (platform specific) source code from the v4.2.1 serialport-rs Github repo <https://github.com/serialport/serialport-rs> (master branch).

Linux specific read() behavior =>

For *Linux*, a positive (> 0) timeout value is the period of time that **read()** blocks when no incoming data has arrived. If at least one (1) byte of data is received and available in the read's internal buffer, whether before initial entry to **read()** or any time thereafter prior to the timeout period expiring, **read()** returns immediately (at that instance) with the available data. If no received data is available at the conclusion of the timeout period, then **read()** returns at that time with a timeout error (and no data). The Linux **read()** returns the available data as soon as its available (event on initial entry), with the returned data count (len) only limited in size by the **read()** invocation's buffer argument length. With a read timeout setting of 0, the Linux **read()** returns with whatever data (if any) is available on entry, or none, but always return immediately, with or without data. When **read()** returns with no data (because none is available), it returns a timeout error indication. Please note that the Linux **read()** non-zero timeout setting does not result in a **read()** internally extending its blocked time period waiting for *additional* data to arrive in order to fully satisfy the requested data length, but rather only extends its wait when no data at all has been received.

Windows specific read() behavior =>

The Window's **read()** for v4.2.1 behaves differently in various respects to the Linux implementation.

On Windows when the timeout value is non-zero (> 0), then the **read()** behaves only roughly similar to Linux, and varies in a couple of measurable aspects.

- a. The Windows **read()** waits for the entire timeout period while the entire requested read byte count (buffer size) is not yet available, and returns only at the conclusion of the **read()** timeout period with the lesser data amount that is available at conclusion of the timeout period - but no timeout occurs as long as at least one byte is returned. The Window's **read()** returns sooner than the timeout period *only if* the read's entire requested data count (the supplied buffer size) is received prior to the conclusion of the read timeout period.
- b. The second difference from the Linux **read()** is more consequential to applications. When the Window's read timeout set-point value is 0, the **read()** blocks until ALL requested bytes (per the supplied buffer's length) become available. Essentially a 0 (timeout) set-point in Windows operates as an infinite timeout period in order to satisfy the read request length. To be clear, in this scenario a 0 timeout setpoint and having less than the read buffer's length of data arriving, the **read()** invocation blocks (forever) and never times-out.

As a practitioner's note regarding the Windows **read()** operation, with smaller read timeout settings in the range from 1 and 15 ms, a **read()** invocation can be randomly extended in time by up to ~15 to 20 ms prior to returning with already received data, or a timeout indication if no data has arrived within the timeout period. While this is certainly not ideal, by all account it's something Microsoft is fully aware of and offers no solution to. Their documentation indicates this relates to the Windows OS API specification that its sleep timers have a published resolution of no less than ~16 milliseconds. This random delay occurs with its native API Readfile() method, which is internally used by the crate's **read()** trait method, and is typically only a significant timing hindrance if one is attempting to write applications which must deterministically (hard real-time) respond to incoming serial data with repeatable timing accuracies in the sub (<) 100 ms range. These types of application timing requirements are generally not suitable for Windows as a target computer, for this and several other reasons. It is well known that Windows is *not* a real-time operating system!

Nevertheless, there are (many) successful and reliable industrial control applications which are quite effective using RS-232 serial communication connections while running on modern Windows high-performance multi-core CPU based computer system. They are effective and demonstrate high percentage repeatability with timer accuracies in the ~100 ms (and above) time region. The Linux OS can of course offer performance which is somewhat better than Windows, using its **-rt** (kernel option) and its **pthread** real-time priority threads - but I'm unaware if Rust threads on Linux can 'out of the box' utilize this Linux platform specific threading feature.

'receive_timing_info' test example data log snippets for three (3) command argument scenarios with read timeouts.

Next are three (3) generated Windows test logs with parameters which intentionally induce read timeouts. Note that each log file's second text line entry indicates the run-time supplied command line argument values which were provided.

Note that only the initial (first) half cycle of each of these test runs was undertaken, since the induced read timeout (induced by specifying an `-rxlen` and `-txlen` parameter pair where `rxlen > txlen`) resulted in a 'transfer stalled' timeout error during its first half cycle.

```
2023-07-16 19:11:59.2347468 INFO receive_timing_info - 'receive_timing_info' cross platform dual RS-232 port null modem cable connected rcv+xmt+timeout test and characterization tool: v1.0
2023-07-16 19:11:59.2351164 INFO receive_timing_info - Test setup: Platform='windows', Baud=200000, rxtmo=50 ms, posttxdelays=100 ms, xfrstalledtmo=1000 ms, txlen=1, rxlen=2, repeat=1, fulldbgs=false
2023-07-16 19:11:59.2378638 INFO receive_timing_info - Test Logfile Name: 'D:\Users\ricej\windows_receive_timing_info.txt'
2023-07-16 19:11:59.29698 INFO receive_timing_info -
2023-07-16 19:11:59.2969872 INFO receive_timing_info - ** Start of cycle 1. **
2023-07-16 19:11:59.2969881 INFO receive_timing_info - Cycle 1 first phase -> Rx port = 'COM7', Tx port = 'COM6' .
2023-07-16 19:11:59.65374 INFO receive_timing_info - txport.write() sent 1 bytes while blocked for 548 us. Read() invoked 100412 us after write(), rxport.read(2) returned 1 bytes while blocked for 50255 us.
2023-07-16 19:11:59.7129727 INFO receive_timing_info - Read() invoked 150719 us after write(), rxport.read(1) returned 0 bytes while blocked for 59176 us. Rcv timeout.
2023-07-16 19:11:59.7629789 INFO receive_timing_info - Read() invoked 209949 us after write(), rxport.read(1) returned 0 bytes while blocked for 49959 us. Rcv timeout.
2023-07-16 19:11:59.8131091 INFO receive_timing_info - Read() invoked 259953 us after write(), rxport.read(1) returned 0 bytes while blocked for 50050 us. Rcv timeout.
2023-07-16 19:11:59.8729912 INFO receive_timing_info - Read() invoked 310088 us after write(), rxport.read(1) returned 0 bytes while blocked for 59832 us. Rcv timeout.
2023-07-16 19:11:59.923074 INFO receive_timing_info - Read() invoked 369984 us after write(), rxport.read(1) returned 0 bytes while blocked for 50024 us. Rcv timeout.
2023-07-16 19:11:59.9730056 INFO receive_timing_info - Read() invoked 420051 us after write(), rxport.read(1) returned 0 bytes while blocked for 49882 us. Rcv timeout.
2023-07-16 19:12:00.0301661 INFO receive_timing_info - Read() invoked 469980 us after write(), rxport.read(1) returned 0 bytes while blocked for 57127 us. Rcv timeout.
2023-07-16 19:12:00.0802684 INFO receive_timing_info - Read() invoked 527131 us after write(), rxport.read(1) returned 0 bytes while blocked for 50065 us. Rcv timeout.
2023-07-16 19:12:00.1401687 INFO receive_timing_info - Read() invoked 577248 us after write(), rxport.read(1) returned 0 bytes while blocked for 59835 us. Rcv timeout.
2023-07-16 19:12:00.190347 INFO receive_timing_info - Read() invoked 637145 us after write(), rxport.read(1) returned 0 bytes while blocked for 50135 us. Rcv timeout.
2023-07-16 19:12:00.2403654 INFO receive_timing_info - Read() invoked 687319 us after write(), rxport.read(1) returned 0 bytes while blocked for 49979 us. Rcv timeout.
2023-07-16 19:12:00.2992358 INFO receive_timing_info - Read() invoked 737337 us after write(), rxport.read(1) returned 0 bytes while blocked for 58831 us. Rcv timeout.
2023-07-16 19:12:00.3492649 INFO receive_timing_info - Read() invoked 796206 us after write(), rxport.read(1) returned 0 bytes while blocked for 49992 us. Rcv timeout.
2023-07-16 19:12:00.3992507 INFO receive_timing_info - Read() invoked 846241 us after write(), rxport.read(1) returned 0 bytes while blocked for 49941 us. Rcv timeout.
2023-07-16 19:12:00.458996 INFO receive_timing_info - Read() invoked 896227 us after write(), rxport.read(1) returned 0 bytes while blocked for 59714 us. Rcv timeout.
2023-07-16 19:12:00.5091288 INFO receive_timing_info - Read() invoked 955960 us after write(), rxport.read(1) returned 0 bytes while blocked for 50082 us. Rcv timeout.
2023-07-16 19:12:00.5690609 INFO receive_timing_info - Read() invoked 1006104 us after write(), rxport.read(1) returned 0 bytes while blocked for 59874 us. Rcv timeout.
2023-07-16 19:12:00.6191622 INFO receive_timing_info - Read() invoked 1066021 us after write(), rxport.read(1) returned 0 bytes while blocked for 50077 us. Rcv timeout.
2023-07-16 19:12:00.6696142 INFO receive_timing_info - Read() invoked 1116135 us after write(), rxport.read(1) returned 0 bytes while blocked for 50413 us. Rcv timeout.
2023-07-16 19:12:00.6696732 INFO receive_timing_info -
TRANSFER STALLED TIMEOUT ERROR: 'rxport::read()' repeatedly timed-out without receiving its requested incoming data. If not induced, inspect-verify the serial connections. Aborting.
```

The second Windows log example:

```
2023-07-16 19:13:30.8416232 INFO receive_timing_info - 'receive_timing_info' cross platform dual RS-232 port null modem cable connected rcv+xmt+timeout test and characterization tool: v1.0
2023-07-16 19:13:30.84176 INFO receive_timing_info - Test setup: Platform='windows', Baud=200000, rxtmo=1 ms, posttxdelays=0 ms, xfrstalledtmo=1000 ms, txlen=1, rxlen=2, repeat=1, fulldbgs=false
2023-07-16 19:13:30.8458403 INFO receive_timing_info - Test Logfile Name: 'D:\Users\ricej\windows_receive_timing_info.txt'
2023-07-16 19:13:30.8690723 INFO receive_timing_info -
2023-07-16 19:13:30.8690797 INFO receive_timing_info - ** Start of cycle 1. **
2023-07-16 19:13:30.8690811 INFO receive_timing_info - Cycle 1 first phase -> Rx port = 'COM7', Tx port = 'COM6' .
2023-07-16 19:13:31.0837107 INFO receive_timing_info - txport.write() sent 1 bytes while blocked for 427 us. Read() invoked 25 us after write(), rxport.read(2) returned 0 bytes while blocked for 9449 us. Rcv timeout.
```


2023-07-16 19:13:31.0935963 INFO receive_timing_info -
2023-07-16 19:13:31.1037244 INFO receive_timing_info -
2023-07-16 19:13:31.1137371 INFO receive_timing_info -
2023-07-16 19:13:31.1237386 INFO receive_timing_info -
2023-07-16 19:13:31.1337384 INFO receive_timing_info -
2023-07-16 19:13:31.1435955 INFO receive_timing_info -
2023-07-16 19:13:31.1537175 INFO receive_timing_info -
2023-07-16 19:13:31.1637297 INFO receive_timing_info -
2023-07-16 19:13:31.1737385 INFO receive_timing_info -
2023-07-16 19:13:31.1837062 INFO receive_timing_info -
2023-07-16 19:13:31.1936502 INFO receive_timing_info -
2023-07-16 19:13:31.2037372 INFO receive_timing_info -
2023-07-16 19:13:31.2137275 INFO receive_timing_info -
2023-07-16 19:13:31.223706 INFO receive_timing_info -
2023-07-16 19:13:31.2337312 INFO receive_timing_info -
2023-07-16 19:13:31.2436519 INFO receive_timing_info -
2023-07-16 19:13:31.2537485 INFO receive_timing_info -
2023-07-16 19:13:31.263729 INFO receive_timing_info -
2023-07-16 19:13:31.2760295 INFO receive_timing_info -
2023-07-16 19:13:31.2860841 INFO receive_timing_info -
2023-07-16 19:13:31.2961383 INFO receive_timing_info -
2023-07-16 19:13:31.3143309 INFO receive_timing_info -
2023-07-16 19:13:31.3243362 INFO receive_timing_info -
2023-07-16 19:13:31.3343353 INFO receive_timing_info -
2023-07-16 19:13:31.3442805 INFO receive_timing_info -
2023-07-16 19:13:31.3543142 INFO receive_timing_info -
2023-07-16 19:13:31.3643089 INFO receive_timing_info -
2023-07-16 19:13:31.3743395 INFO receive_timing_info -
2023-07-16 19:13:31.3842602 INFO receive_timing_info -
2023-07-16 19:13:31.3959133 INFO receive_timing_info -
2023-07-16 19:13:31.4146801 INFO receive_timing_info -
2023-07-16 19:13:31.4245156 INFO receive_timing_info -
2023-07-16 19:13:31.4346106 INFO receive_timing_info -
2023-07-16 19:13:31.4446121 INFO receive_timing_info -
2023-07-16 19:13:31.4546299 INFO receive_timing_info -
2023-07-16 19:13:31.464612 INFO receive_timing_info -
2023-07-16 19:13:31.4746493 INFO receive_timing_info -
2023-07-16 19:13:31.4846173 INFO receive_timing_info -
2023-07-16 19:13:31.49463 INFO receive_timing_info -
2023-07-16 19:13:31.5045199 INFO receive_timing_info -
2023-07-16 19:13:31.5146423 INFO receive_timing_info -
2023-07-16 19:13:31.524661 INFO receive_timing_info -
2023-07-16 19:13:31.5346001 INFO receive_timing_info -
2023-07-16 19:13:31.5445119 INFO receive_timing_info -
2023-07-16 19:13:31.5546218 INFO receive_timing_info -
2023-07-16 19:13:31.5646134 INFO receive_timing_info -
2023-07-16 19:13:31.5746458 INFO receive_timing_info -
2023-07-16 19:13:31.5845786 INFO receive_timing_info -
2023-07-16 19:13:31.5946506 INFO receive_timing_info -
2023-07-16 19:13:31.6046229 INFO receive_timing_info -
2023-07-16 19:13:31.6146285 INFO receive_timing_info -

Read() invoked 9520 us after write(), rxport.read(2) returned 1 bytes while blocked for 9827 us.
Read() invoked 19400 us after write(), rxport.read(1) returned 0 bytes while blocked for 10087 us. Rcv timeout.
Read() invoked 29555 us after write(), rxport.read(1) returned 0 bytes while blocked for 9945 us. Rcv timeout.
Read() invoked 39547 us after write(), rxport.read(1) returned 0 bytes while blocked for 9953 us. Rcv timeout.
Read() invoked 49549 us after write(), rxport.read(1) returned 0 bytes while blocked for 9951 us. Rcv timeout.
Read() invoked 59552 us after write(), rxport.read(1) returned 0 bytes while blocked for 9806 us. Rcv timeout.
Read() invoked 69407 us after write(), rxport.read(1) returned 0 bytes while blocked for 10074 us. Rcv timeout.
Read() invoked 79529 us after write(), rxport.read(1) returned 0 bytes while blocked for 9966 us. Rcv timeout.
Read() invoked 89542 us after write(), rxport.read(1) returned 0 bytes while blocked for 9963 us. Rcv timeout.
Read() invoked 99549 us after write(), rxport.read(1) returned 0 bytes while blocked for 9922 us. Rcv timeout.
Read() invoked 109518 us after write(), rxport.read(1) returned 0 bytes while blocked for 9898 us. Rcv timeout.
Read() invoked 119456 us after write(), rxport.read(1) returned 0 bytes while blocked for 10045 us. Rcv timeout.
Read() invoked 129550 us after write(), rxport.read(1) returned 0 bytes while blocked for 9944 us. Rcv timeout.
Read() invoked 139539 us after write(), rxport.read(1) returned 0 bytes while blocked for 9932 us. Rcv timeout.
Read() invoked 149519 us after write(), rxport.read(1) returned 0 bytes while blocked for 9978 us. Rcv timeout.
Read() invoked 159544 us after write(), rxport.read(1) returned 0 bytes while blocked for 9870 us. Rcv timeout.
Read() invoked 169468 us after write(), rxport.read(1) returned 0 bytes while blocked for 10046 us. Rcv timeout.
Read() invoked 179559 us after write(), rxport.read(1) returned 0 bytes while blocked for 9937 us. Rcv timeout.
Read() invoked 189541 us after write(), rxport.read(1) returned 0 bytes while blocked for 12259 us. Rcv timeout.
Read() invoked 201835 us after write(), rxport.read(1) returned 0 bytes while blocked for 10016 us. Rcv timeout.
Read() invoked 211896 us after write(), rxport.read(1) returned 0 bytes while blocked for 10006 us. Rcv timeout.
Read() invoked 221949 us after write(), rxport.read(1) returned 0 bytes while blocked for 18144 us. Rcv timeout.
Read() invoked 240141 us after write(), rxport.read(1) returned 0 bytes while blocked for 9960 us. Rcv timeout.
Read() invoked 250148 us after write(), rxport.read(1) returned 0 bytes while blocked for 9948 us. Rcv timeout.
Read() invoked 260143 us after write(), rxport.read(1) returned 0 bytes while blocked for 9903 us. Rcv timeout.
Read() invoked 270112 us after write(), rxport.read(1) returned 0 bytes while blocked for 9969 us. Rcv timeout.
Read() invoked 280127 us after write(), rxport.read(1) returned 0 bytes while blocked for 9949 us. Rcv timeout.
Read() invoked 290121 us after write(), rxport.read(1) returned 0 bytes while blocked for 9984 us. Rcv timeout.
Read() invoked 300151 us after write(), rxport.read(1) returned 0 bytes while blocked for 9871 us. Rcv timeout.
Read() invoked 310072 us after write(), rxport.read(1) returned 0 bytes while blocked for 11609 us. Rcv timeout.
Read() invoked 321722 us after write(), rxport.read(1) returned 0 bytes while blocked for 18720 us. Rcv timeout.
Read() invoked 340484 us after write(), rxport.read(1) returned 0 bytes while blocked for 9796 us. Rcv timeout.
Read() invoked 350328 us after write(), rxport.read(1) returned 0 bytes while blocked for 10048 us. Rcv timeout.
Read() invoked 360415 us after write(), rxport.read(1) returned 0 bytes while blocked for 9962 us. Rcv timeout.
Read() invoked 370427 us after write(), rxport.read(1) returned 0 bytes while blocked for 9969 us. Rcv timeout.
Read() invoked 380441 us after write(), rxport.read(1) returned 0 bytes while blocked for 9937 us. Rcv timeout.
Read() invoked 390424 us after write(), rxport.read(1) returned 0 bytes while blocked for 9989 us. Rcv timeout.
Read() invoked 400462 us after write(), rxport.read(1) returned 0 bytes while blocked for 9920 us. Rcv timeout.
Read() invoked 410434 us after write(), rxport.read(1) returned 0 bytes while blocked for 9962 us. Rcv timeout.
Read() invoked 420442 us after write(), rxport.read(1) returned 0 bytes while blocked for 9842 us. Rcv timeout.
Read() invoked 430334 us after write(), rxport.read(1) returned 0 bytes while blocked for 10075 us. Rcv timeout.
Read() invoked 440453 us after write(), rxport.read(1) returned 0 bytes while blocked for 9974 us. Rcv timeout.
Read() invoked 450494 us after write(), rxport.read(1) returned 0 bytes while blocked for 9869 us. Rcv timeout.
Read() invoked 460414 us after write(), rxport.read(1) returned 0 bytes while blocked for 9864 us. Rcv timeout.
Read() invoked 470323 us after write(), rxport.read(1) returned 0 bytes while blocked for 10064 us. Rcv timeout.
Read() invoked 480432 us after write(), rxport.read(1) returned 0 bytes while blocked for 9945 us. Rcv timeout.
Read() invoked 490426 us after write(), rxport.read(1) returned 0 bytes while blocked for 9987 us. Rcv timeout.
Read() invoked 500456 us after write(), rxport.read(1) returned 0 bytes while blocked for 9869 us. Rcv timeout.
Read() invoked 510385 us after write(), rxport.read(1) returned 0 bytes while blocked for 10027 us. Rcv timeout.
Read() invoked 520465 us after write(), rxport.read(1) returned 0 bytes while blocked for 9922 us. Rcv timeout.
Read() invoked 530438 us after write(), rxport.read(1) returned 0 bytes while blocked for 9959 us. Rcv timeout.

2023-07-16 19:13:31.6246623 INFO receive_timing_info -
2023-07-16 19:13:31.6345675 INFO receive_timing_info -
2023-07-16 19:13:31.6446325 INFO receive_timing_info -
2023-07-16 19:13:31.6546681 INFO receive_timing_info -
2023-07-16 19:13:31.6645206 INFO receive_timing_info -
2023-07-16 19:13:31.6746493 INFO receive_timing_info -
2023-07-16 19:13:31.6845833 INFO receive_timing_info -
2023-07-16 19:13:31.6946451 INFO receive_timing_info -
2023-07-16 19:13:31.7044569 INFO receive_timing_info -
2023-07-16 19:13:31.7146347 INFO receive_timing_info -
2023-07-16 19:13:31.7246317 INFO receive_timing_info -
2023-07-16 19:13:31.7347149 INFO receive_timing_info -
2023-07-16 19:13:31.74461 INFO receive_timing_info -
2023-07-16 19:13:31.7545724 INFO receive_timing_info -
2023-07-16 19:13:31.7646463 INFO receive_timing_info -
2023-07-16 19:13:31.7746377 INFO receive_timing_info -
2023-07-16 19:13:31.7844717 INFO receive_timing_info -
2023-07-16 19:13:31.7946455 INFO receive_timing_info -
2023-07-16 19:13:31.8046542 INFO receive_timing_info -
2023-07-16 19:13:31.8145803 INFO receive_timing_info -
2023-07-16 19:13:31.8246227 INFO receive_timing_info -
2023-07-16 19:13:31.8345487 INFO receive_timing_info -
2023-07-16 19:13:31.8445683 INFO receive_timing_info -
2023-07-16 19:13:31.8562955 INFO receive_timing_info -
2023-07-16 19:13:31.8664263 INFO receive_timing_info -
2023-07-16 19:13:31.8763461 INFO receive_timing_info -
2023-07-16 19:13:31.8864335 INFO receive_timing_info -
2023-07-16 19:13:31.8962505 INFO receive_timing_info -
2023-07-16 19:13:31.9150854 INFO receive_timing_info -
2023-07-16 19:13:31.9250768 INFO receive_timing_info -
2023-07-16 19:13:31.9349086 INFO receive_timing_info -
2023-07-16 19:13:31.9450669 INFO receive_timing_info -
2023-07-16 19:13:31.9550776 INFO receive_timing_info -
2023-07-16 19:13:31.9650426 INFO receive_timing_info -
2023-07-16 19:13:31.9750956 INFO receive_timing_info -
2023-07-16 19:13:31.9849724 INFO receive_timing_info -
2023-07-16 19:13:31.9949931 INFO receive_timing_info -
2023-07-16 19:13:32.0050653 INFO receive_timing_info -
2023-07-16 19:13:32.0149805 INFO receive_timing_info -
2023-07-16 19:13:32.0250305 INFO receive_timing_info -
2023-07-16 19:13:32.0350116 INFO receive_timing_info -
2023-07-16 19:13:32.0450704 INFO receive_timing_info -
2023-07-16 19:13:32.0549696 INFO receive_timing_info -
2023-07-16 19:13:32.0649347 INFO receive_timing_info -
2023-07-16 19:13:32.074916 INFO receive_timing_info -
2023-07-16 19:13:32.0849049 INFO receive_timing_info -
2023-07-16 19:13:32.0949135 INFO receive_timing_info -
2023-07-16 19:13:32.0949308 INFO receive_timing_info -

TRANSFER STALLED TIMEOUT ERROR: 'rxport::read()' repeatedly timed-out without receiving its requested incoming data. If not induced, inspect+verify the serial connections. Aborting.

Read() invoked 540438 us after write(), rxport.read(1) returned 0 bytes while blocked for 9964 us. Rcv timeout.
Read() invoked 550476 us after write(), rxport.read(1) returned 0 bytes while blocked for 9853 us. Rcv timeout.
Read() invoked 560379 us after write(), rxport.read(1) returned 0 bytes while blocked for 10019 us. Rcv timeout.
Read() invoked 570443 us after write(), rxport.read(1) returned 0 bytes while blocked for 9987 us. Rcv timeout.
Read() invoked 580486 us after write(), rxport.read(1) returned 0 bytes while blocked for 9799 us. Rcv timeout.
Read() invoked 590332 us after write(), rxport.read(1) returned 0 bytes while blocked for 10083 us. Rcv timeout.
Read() invoked 600462 us after write(), rxport.read(1) returned 0 bytes while blocked for 9889 us. Rcv timeout.
Read() invoked 610393 us after write(), rxport.read(1) returned 0 bytes while blocked for 10019 us. Rcv timeout.
Read() invoked 620456 us after write(), rxport.read(1) returned 0 bytes while blocked for 9755 us. Rcv timeout.
Read() invoked 630267 us after write(), rxport.read(1) returned 0 bytes while blocked for 10135 us. Rcv timeout.
Read() invoked 640446 us after write(), rxport.read(1) returned 0 bytes while blocked for 9952 us. Rcv timeout.
Read() invoked 650443 us after write(), rxport.read(1) returned 0 bytes while blocked for 10037 us. Rcv timeout.
Read() invoked 660524 us after write(), rxport.read(1) returned 0 bytes while blocked for 9847 us. Rcv timeout.
Read() invoked 670422 us after write(), rxport.read(1) returned 0 bytes while blocked for 9911 us. Rcv timeout.
Read() invoked 680383 us after write(), rxport.read(1) returned 0 bytes while blocked for 10028 us. Rcv timeout.
Read() invoked 690457 us after write(), rxport.read(1) returned 0 bytes while blocked for 9948 us. Rcv timeout.
Read() invoked 700448 us after write(), rxport.read(1) returned 0 bytes while blocked for 9784 us. Rcv timeout.
Read() invoked 710282 us after write(), rxport.read(1) returned 0 bytes while blocked for 10129 us. Rcv timeout.
Read() invoked 720458 us after write(), rxport.read(1) returned 0 bytes while blocked for 9957 us. Rcv timeout.
Read() invoked 730471 us after write(), rxport.read(1) returned 0 bytes while blocked for 9870 us. Rcv timeout.
Read() invoked 740391 us after write(), rxport.read(1) returned 0 bytes while blocked for 9998 us. Rcv timeout.
Read() invoked 750434 us after write(), rxport.read(1) returned 0 bytes while blocked for 9876 us. Rcv timeout.
Read() invoked 760363 us after write(), rxport.read(1) returned 0 bytes while blocked for 9988 us. Rcv timeout.
Read() invoked 770367 us after write(), rxport.read(1) returned 0 bytes while blocked for 11700 us. Rcv timeout.
Read() invoked 782097 us after write(), rxport.read(1) returned 0 bytes while blocked for 10095 us. Rcv timeout.
Read() invoked 792237 us after write(), rxport.read(1) returned 0 bytes while blocked for 9871 us. Rcv timeout.
Read() invoked 802153 us after write(), rxport.read(1) returned 0 bytes while blocked for 10048 us. Rcv timeout.
Read() invoked 812242 us after write(), rxport.read(1) returned 0 bytes while blocked for 9779 us. Rcv timeout.
Read() invoked 822056 us after write(), rxport.read(1) returned 0 bytes while blocked for 18794 us. Rcv timeout.
Read() invoked 840901 us after write(), rxport.read(1) returned 0 bytes while blocked for 9943 us. Rcv timeout.
Read() invoked 850888 us after write(), rxport.read(1) returned 0 bytes while blocked for 9783 us. Rcv timeout.
Read() invoked 860720 us after write(), rxport.read(1) returned 0 bytes while blocked for 10108 us. Rcv timeout.
Read() invoked 870873 us after write(), rxport.read(1) returned 0 bytes while blocked for 9967 us. Rcv timeout.
Read() invoked 880889 us after write(), rxport.read(1) returned 0 bytes while blocked for 9920 us. Rcv timeout.
Read() invoked 890853 us after write(), rxport.read(1) returned 0 bytes while blocked for 9978 us. Rcv timeout.
Read() invoked 900902 us after write(), rxport.read(1) returned 0 bytes while blocked for 9835 us. Rcv timeout.
Read() invoked 910784 us after write(), rxport.read(1) returned 0 bytes while blocked for 9976 us. Rcv timeout.
Read() invoked 920798 us after write(), rxport.read(1) returned 0 bytes while blocked for 10033 us. Rcv timeout.
Read() invoked 930876 us after write(), rxport.read(1) returned 0 bytes while blocked for 9870 us. Rcv timeout.
Read() invoked 940788 us after write(), rxport.read(1) returned 0 bytes while blocked for 10008 us. Rcv timeout.
Read() invoked 950839 us after write(), rxport.read(1) returned 0 bytes while blocked for 9940 us. Rcv timeout.
Read() invoked 960822 us after write(), rxport.read(1) returned 0 bytes while blocked for 9988 us. Rcv timeout.
Read() invoked 970876 us after write(), rxport.read(1) returned 0 bytes while blocked for 9862 us. Rcv timeout.
Read() invoked 980779 us after write(), rxport.read(1) returned 0 bytes while blocked for 9928 us. Rcv timeout.
Read() invoked 990727 us after write(), rxport.read(1) returned 0 bytes while blocked for 9973 us. Rcv timeout.
Read() invoked 1000709 us after write(), rxport.read(1) returned 0 bytes while blocked for 9977 us. Rcv timeout.
Read() invoked 1010697 us after write(), rxport.read(1) returned 0 bytes while blocked for 10000 us. Rcv timeout.

Here's the third Windows example log, which demonstrates the indefinite blocking issue with read() when the read timeout setpoint is 0 and all requested receive data doesn't arrive. It also has the –fulldbg flag enabled.

```
2023-07-16 19:17:03.334684 INFO receive_timing_info - 'receive_timing_info' cross platform dual RS-232 port null modem cable connected rcv+xmt+timeout test and characterization tool: v1.0
2023-07-16 19:17:03.3348747 INFO receive_timing_info - Test setup: Platform='windows', Baud=200000, rxtmo=0 ms, posttxdelays=0 ms, xfrstalledtmo=1000 ms, txlen=1, rxlen=2, repeat=1, fulldbg=true
2023-07-16 19:17:03.3377227 INFO receive_timing_info - Test Logfile Name: 'D:\Users\ricej\windows_receive_timing_info.txt'
2023-07-16 19:17:03.4205881 INFO receive_timing_info -
2023-07-16 19:17:03.4205962 INFO receive_timing_info - ** Start of cycle 1. **
2023-07-16 19:17:03.4205977 INFO receive_timing_info - Cycle 1 first phase -> Rx port = 'COM7', Tx port = 'COM6' .
2023-07-16 19:17:03.62881 INFO receive_timing_info - Enter write(txbuf[0..1]).
2023-07-16 19:17:03.6291673 INFO receive_timing_info - Return from write().
2023-07-16 19:17:03.6292021 INFO receive_timing_info - Enter read(rxbuf[0..2]).
```

The above log-file content was as shown following the test application being blocked/suspended for approximately two (2) minutes, at which time I terminated the test application from the keyboard.

Next are the equivalent Linux logs for the same three (3) test invocation argument sets – generated from Ubuntu Linux executing in a VMWare guest on the same laptop. The first log =>

```
2023-07-16 16:43:14.272409174 INFO receive_timing_info - 'receive_timing_info' cross platform dual RS-232 port null modem cable connected rcv+xmt+timeout test and characterization tool: v1.0
2023-07-16 16:43:14.272426885 INFO receive_timing_info - Test setup: Platform='linux', Baud=200000, rxtmo=50 ms, posttxdelays=100 ms, xfrstalledtmo=1000 ms, txlen=1, rxlen=2, repeat=1, fulldbg=false
2023-07-16 16:43:14.272434667 INFO receive_timing_info - Test Logfile Name: '/home/ricej/ubuntu_receive_timing_info.txt'
2023-07-16 16:43:14.895900214 INFO receive_timing_info -
2023-07-16 16:43:14.895949482 INFO receive_timing_info - ** Start of cycle 1. **
2023-07-16 16:43:14.89595447 INFO receive_timing_info - Cycle 1 first phase -> Rx port = '/dev/ttyUSB1', Tx port = '/dev/ttyUSB0' .
2023-07-16 16:43:15.198883731 INFO receive_timing_info - txport.write() sent 1 bytes while blocked for 2425 us. Read() invoked 100269 us after write(), rxport.read(2) returned 1 bytes while blocked for 17 us.
2023-07-16 16:43:15.249000315 INFO receive_timing_info - Read() invoked 100344 us after write(), rxport.read(1) returned 0 bytes while blocked for 50064 us. Rcv timeout.
2023-07-16 16:43:15.299208892 INFO receive_timing_info - Read() invoked 150429 us after write(), rxport.read(1) returned 0 bytes while blocked for 50186 us. Rcv timeout.
2023-07-16 16:43:15.350425949 INFO receive_timing_info - Read() invoked 200662 us after write(), rxport.read(1) returned 0 bytes while blocked for 51169 us. Rcv timeout.
2023-07-16 16:43:15.400619098 INFO receive_timing_info - Read() invoked 251881 us after write(), rxport.read(1) returned 0 bytes while blocked for 50143 us. Rcv timeout.
2023-07-16 16:43:15.451641702 INFO receive_timing_info - Read() invoked 302069 us after write(), rxport.read(1) returned 0 bytes while blocked for 50977 us. Rcv timeout.
2023-07-16 16:43:15.502401107 INFO receive_timing_info - Read() invoked 353102 us after write(), rxport.read(1) returned 0 bytes while blocked for 50699 us. Rcv timeout.
2023-07-16 16:43:15.553152011 INFO receive_timing_info - Read() invoked 403870 us after write(), rxport.read(1) returned 0 bytes while blocked for 50684 us. Rcv timeout.
2023-07-16 16:43:15.603731655 INFO receive_timing_info - Read() invoked 454621 us after write(), rxport.read(1) returned 0 bytes while blocked for 50516 us. Rcv timeout.
2023-07-16 16:43:15.654415799 INFO receive_timing_info - Read() invoked 505188 us after write(), rxport.read(1) returned 0 bytes while blocked for 50633 us. Rcv timeout.
2023-07-16 16:43:15.705176217 INFO receive_timing_info - Read() invoked 555874 us after write(), rxport.read(1) returned 0 bytes while blocked for 50707 us. Rcv timeout.
2023-07-16 16:43:15.756190409 INFO receive_timing_info - Read() invoked 606634 us after write(), rxport.read(1) returned 0 bytes while blocked for 50962 us. Rcv timeout.
2023-07-16 16:43:15.806492464 INFO receive_timing_info - Read() invoked 657645 us after write(), rxport.read(1) returned 0 bytes while blocked for 50252 us. Rcv timeout.
2023-07-16 16:43:15.857302959 INFO receive_timing_info - Read() invoked 707978 us after write(), rxport.read(1) returned 0 bytes while blocked for 50730 us. Rcv timeout.
2023-07-16 16:43:15.908021805 INFO receive_timing_info - Read() invoked 758762 us after write(), rxport.read(1) returned 0 bytes while blocked for 50666 us. Rcv timeout.
2023-07-16 16:43:15.958705464 INFO receive_timing_info - Read() invoked 809474 us after write(), rxport.read(1) returned 0 bytes while blocked for 50637 us. Rcv timeout.
2023-07-16 16:43:16.009378698 INFO receive_timing_info - Read() invoked 860161 us after write(), rxport.read(1) returned 0 bytes while blocked for 50621 us. Rcv timeout.
2023-07-16 16:43:16.059899261 INFO receive_timing_info - Read() invoked 910848 us after write(), rxport.read(1) returned 0 bytes while blocked for 50456 us. Rcv timeout.
2023-07-16 16:43:16.110283419 INFO receive_timing_info - Read() invoked 961356 us after write(), rxport.read(1) returned 0 bytes while blocked for 50332 us. Rcv timeout.
2023-07-16 16:43:16.160563977 INFO receive_timing_info - Read() invoked 1011715 us after write(), rxport.read(1) returned 0 bytes while blocked for 50254 us. Rcv timeout.
2023-07-16 16:43:16.211238623 INFO receive_timing_info - Read() invoked 1062019 us after write(), rxport.read(1) returned 0 bytes while blocked for 50625 us. Rcv timeout.
2023-07-16 16:43:16.211282556 INFO receive_timing_info -
TRANSFER STALLED TIMEOUT ERROR: 'rxport::read()' repeatedly timed-out without receiving its requested incoming data. If not induced, inspect+verify the serial connections. Aborting.
```

Here's the second Linux log example. This full log is considerably lengthier than the equivalent Windows log, due to Linux more responsive (i.e. faster) **read()** timeout returns. For brevity I've excluded many uninteresting intermediate **read()** log entries soon after the initial byte is read, and resume with the final reads leading up to the terminating transfer stalled timeout error - around 1 second after the initial **read()** following the earlier matching **write()**.

```
2023-07-16 16:36:48.733217257 INFO receive_timing_info - 'receive_timing_info' cross platform dual RS-232 port null modem cable connected rcv+xmt+timeout test and characterization tool: v1.0
2023-07-16 16:36:48.733246792 INFO receive_timing_info - Test setup: Platform='linux', Baud=200000, rx_tmo=1 ms, postxdelays=50 ms, xfrstalledtmo=1000 ms, txlen=1, rxlen=2, repeat=1, fulldb=false
2023-07-16 16:36:48.733254245 INFO receive_timing_info - Test Logfile Name: '/home/ricej/ubuntu_receive_timing_info.txt'
2023-07-16 16:36:49.716092489 INFO receive_timing_info -
2023-07-16 16:36:49.716142723 INFO receive_timing_info - ** Start of cycle 1. **
2023-07-16 16:36:49.716146655 INFO receive_timing_info - Cycle 1 first phase -> Rx port = '/dev/ttyUSB1', Tx port = '/dev/ttyUSB0'.
2023-07-16 16:36:49.969758513 INFO receive_timing_info - txport.write() sent 1 bytes while blocked for 2773 us. Read() invoked 50299 us after write(), rxport.read(2) returned 1 bytes while blocked for 23 us.
2023-07-16 16:36:49.971226136 INFO receive_timing_info - Read() invoked 50380 us after write(), rxport.read(1) returned 0 bytes while blocked for 1420 us. Rcv timeout.
2023-07-16 16:36:49.972910507 INFO receive_timing_info - Read() invoked 51818 us after write(), rxport.read(1) returned 0 bytes while blocked for 1662 us. Rcv timeout.
2023-07-16 16:36:49.974913961 INFO receive_timing_info - Read() invoked 53501 us after write(), rxport.read(1) returned 0 bytes while blocked for 1989 us. Rcv timeout.
2023-07-16 16:36:49.976142741 INFO receive_timing_info - Read() invoked 55495 us after write(), rxport.read(1) returned 0 bytes while blocked for 1223 us. Rcv timeout.
2023-07-16 16:36:49.977314849 INFO receive_timing_info - Read() invoked 56736 us after write(), rxport.read(1) returned 0 bytes while blocked for 1155 us. Rcv timeout.
2023-07-16 16:36:49.97928537 INFO receive_timing_info - Read() invoked 57904 us after write(), rxport.read(1) returned 0 bytes while blocked for 1957 us. Rcv timeout.
2023-07-16 16:36:49.98031803 INFO receive_timing_info - Read() invoked 59874 us after write(), rxport.read(1) returned 0 bytes while blocked for 1021 us. Rcv timeout.
2023-07-16 16:36:49.982303652 INFO receive_timing_info - Read() invoked 60906 us after write(), rxport.read(1) returned 0 bytes while blocked for 1974 us. Rcv timeout.
2023-07-16 16:36:49.984328611 INFO receive_timing_info - Read() invoked 62892 us after write(), rxport.read(1) returned 0 bytes while blocked for 2013 us. Rcv timeout.
2023-07-16 16:36:49.986306959 INFO receive_timing_info - Read() invoked 64909 us after write(), rxport.read(1) returned 0 bytes while blocked for 1975 us. Rcv timeout.
2023-07-16 16:36:49.988299764 INFO receive_timing_info - Read() invoked 66895 us after write(), rxport.read(1) returned 0 bytes while blocked for 1981 us. Rcv timeout.
2023-07-16 16:36:49.990303784 INFO receive_timing_info - Read() invoked 68887 us after write(), rxport.read(1) returned 0 bytes while blocked for 1993 us. Rcv timeout.
2023-07-16 16:36:49.992307476 INFO receive_timing_info - Read() invoked 70892 us after write(), rxport.read(1) returned 0 bytes while blocked for 1992 us. Rcv timeout.
2023-07-16 16:36:49.99349152 INFO receive_timing_info - Read() invoked 72895 us after write(), rxport.read(1) returned 0 bytes while blocked for 1172 us. Rcv timeout.
2023-07-16 16:36:49.995454175 INFO receive_timing_info - Read() invoked 74083 us after write(), rxport.read(1) returned 0 bytes while blocked for 1947 us. Rcv timeout.
2023-07-16 16:36:49.996552099 INFO receive_timing_info - Read() invoked 76043 us after write(), rxport.read(1) returned 0 bytes while blocked for 1086 us. Rcv timeout.
```

... .. <intentionally omitted uninteresting **read()** log entries ... >

... .. <intentionally omitted uninteresting **read()** log entries ... >

... .. <intentionally omitted uninteresting **read()** log entries ... >

```
2023-07-16 16:36:50.94697694 INFO receive_timing_info - Read() invoked 1026199 us after write(), rxport.read(1) returned 0 bytes while blocked for 1348 us. Rcv timeout.
2023-07-16 16:36:50.948476102 INFO receive_timing_info - Read() invoked 1027599 us after write(), rxport.read(1) returned 0 bytes while blocked for 1448 us. Rcv timeout.
2023-07-16 16:36:50.949956468 INFO receive_timing_info - Read() invoked 1029098 us after write(), rxport.read(1) returned 0 bytes while blocked for 1430 us. Rcv timeout.
2023-07-16 16:36:50.951655698 INFO receive_timing_info - Read() invoked 1030575 us after write(), rxport.read(1) returned 0 bytes while blocked for 1652 us. Rcv timeout.
2023-07-16 16:36:50.953217239 INFO receive_timing_info - Read() invoked 1032274 us after write(), rxport.read(1) returned 0 bytes while blocked for 1515 us. Rcv timeout.
2023-07-16 16:36:50.954952595 INFO receive_timing_info - Read() invoked 1033836 us after write(), rxport.read(1) returned 0 bytes while blocked for 1689 us. Rcv timeout.
2023-07-16 16:36:50.956611695 INFO receive_timing_info - Read() invoked 1035569 us after write(), rxport.read(1) returned 0 bytes while blocked for 1615 us. Rcv timeout.
2023-07-16 16:36:50.958619843 INFO receive_timing_info - Read() invoked 1037231 us after write(), rxport.read(1) returned 0 bytes while blocked for 1961 us. Rcv timeout.
2023-07-16 16:36:50.960044618 INFO receive_timing_info - Read() invoked 1039235 us after write(), rxport.read(1) returned 0 bytes while blocked for 1382 us. Rcv timeout.
2023-07-16 16:36:50.962187024 INFO receive_timing_info - Read() invoked 1041244 us after write(), rxport.read(1) returned 0 bytes while blocked for 1515 us. Rcv timeout.
2023-07-16 16:36:50.963525653 INFO receive_timing_info - Read() invoked 1042804 us after write(), rxport.read(1) returned 0 bytes while blocked for 1294 us. Rcv timeout.
2023-07-16 16:36:50.965264495 INFO receive_timing_info - Read() invoked 1044147 us after write(), rxport.read(1) returned 0 bytes while blocked for 1690 us. Rcv timeout.
2023-07-16 16:36:50.966984191 INFO receive_timing_info - Read() invoked 1045881 us after write(), rxport.read(1) returned 0 bytes while blocked for 1676 us. Rcv timeout.
2023-07-16 16:36:50.968903071 INFO receive_timing_info - Read() invoked 1047599 us after write(), rxport.read(1) returned 0 bytes while blocked for 1876 us. Rcv timeout.
2023-07-16 16:36:50.970892558 INFO receive_timing_info - Read() invoked 1049520 us after write(), rxport.read(1) returned 0 bytes while blocked for 1945 us. Rcv timeout.
2023-07-16 16:36:50.970934005 INFO receive_timing_info -
TRANSFER STALLED TIMEOUT ERROR: 'rxport::read()' repeatedly timed-out without receiving its requested incoming data. If not induced, inspect-verify the serial connections. Aborting.
```

Here's the third Linux example log. Again for brevity this shows the leading section of the full log, followed by the section where the first byte is successfully read(), and third the final section where the transfer timeout error occurs. It that it obviously doesn't incur the indefinite blocking which Windows experiences with a read timeout setpoint of 0 and all requested receive data not arriving. Note also in this Linux test run (as compared to its corresponding Windows run) I didn't enable the **--fulldb** flag, since it merely increases the log size and skews certain timing data and makes it more difficult to interpret the results. To be clear, I've indeed executed this same test argument configuration on Linux with the **--fulldb** flag enabled, and it runs as above, but with many more log entries and skewed timing numbers. You may do so as well, if you're curious about this scenario.

```
2023-07-16 17:33:51.182958831 INFO receive_timing_info - 'receive_timing_info' cross platform dual RS-232 port null modem cable connected rcv+xmt+timeout test and characterization tool: v1.0
2023-07-16 17:33:51.182994242 INFO receive_timing_info - Test setup: Platform='linux', Baud=200000, rxtmo=0 ms, posttxdelays=0 ms, xfrstalledtmo=1000 ms, txlen=1, rxlen=2, repeat=1, fulldb=false
2023-07-16 17:33:51.183009314 INFO receive_timing_info - Test Logfile Name: '/home/ricej/ubuntu_receive_timing_info.txt'
2023-07-16 17:33:51.343628307 INFO receive_timing_info -
2023-07-16 17:33:51.343678069 INFO receive_timing_info - ** Start of cycle 1. **
2023-07-16 17:33:51.343682827 INFO receive_timing_info - Cycle 1 first phase -> Rx port = '/dev/ttyUSB1', Tx port = '/dev/ttyUSB0' .
2023-07-16 17:33:51.54727904 INFO receive_timing_info - txport.write() sent 1 bytes while blocked for 3010 us. Read() invoked 9 us after write(), rxport.read(2) returned 0 bytes while blocked for 12 us. Rcv timeout.
2023-07-16 17:33:51.547330689 INFO receive_timing_info - Read() invoked 68 us after write(), rxport.read(2) returned 0 bytes while blocked for 6 us. Rcv timeout.
2023-07-16 17:33:51.547337022 INFO receive_timing_info - Read() invoked 80 us after write(), rxport.read(2) returned 0 bytes while blocked for 3 us. Rcv timeout.
2023-07-16 17:33:51.547342551 INFO receive_timing_info - Read() invoked 86 us after write(), rxport.read(2) returned 0 bytes while blocked for 3 us. Rcv timeout.
2023-07-16 17:33:51.547348054 INFO receive_timing_info - Read() invoked 92 us after write(), rxport.read(2) returned 0 bytes while blocked for 3 us. Rcv timeout.
2023-07-16 17:33:51.547354325 INFO receive_timing_info - Read() invoked 97 us after write(), rxport.read(2) returned 0 bytes while blocked for 3 us. Rcv timeout.
2023-07-16 17:33:51.547360614 INFO receive_timing_info - Read() invoked 103 us after write(), rxport.read(2) returned 0 bytes while blocked for 3 us. Rcv timeout.
2023-07-16 17:33:51.547366008 INFO receive_timing_info - Read() invoked 110 us after write(), rxport.read(2) returned 0 bytes while blocked for 3 us. Rcv timeout.
2023-07-16 17:33:51.547372565 INFO receive_timing_info - Read() invoked 115 us after write(), rxport.read(2) returned 0 bytes while blocked for 3 us. Rcv timeout.
2023-07-16 17:33:51.547378652 INFO receive_timing_info - Read() invoked 122 us after write(), rxport.read(2) returned 0 bytes while blocked for 3 us. Rcv timeout.
```

... .. <intentionally omitted uninteresting read() log entries ... >
 <intentionally omitted uninteresting read() log entries ... >
 <intentionally omitted uninteresting read() log entries ... >

```
2023-07-16 17:33:51.561370021 INFO receive_timing_info - Read() invoked 14112 us after write(), rxport.read(2) returned 0 bytes while blocked for 3 us. Rcv timeout.
2023-07-16 17:33:51.561376082 INFO receive_timing_info - Read() invoked 14119 us after write(), rxport.read(2) returned 0 bytes while blocked for 3 us. Rcv timeout.
2023-07-16 17:33:51.561382249 INFO receive_timing_info - Read() invoked 14125 us after write(), rxport.read(2) returned 0 bytes while blocked for 3 us. Rcv timeout.
2023-07-16 17:33:51.561388814 INFO receive_timing_info - Read() invoked 14131 us after write(), rxport.read(2) returned 0 bytes while blocked for 3 us. Rcv timeout.
2023-07-16 17:33:51.561394964 INFO receive_timing_info - Read() invoked 14138 us after write(), rxport.read(2) returned 0 bytes while blocked for 3 us. Rcv timeout.
2023-07-16 17:33:51.561664472 INFO receive_timing_info - Read() invoked 14144 us after write(), rxport.read(2) returned 1 bytes while blocked for 264 us.
2023-07-16 17:33:51.561686314 INFO receive_timing_info - Read() invoked 14430 us after write(), rxport.read(1) returned 0 bytes while blocked for 3 us. Rcv timeout.
2023-07-16 17:33:51.561687918 INFO receive_timing_info - Read() invoked 14435 us after write(), rxport.read(1) returned 0 bytes while blocked for 1 us. Rcv timeout.
2023-07-16 17:33:51.561689288 INFO receive_timing_info - Read() invoked 14437 us after write(), rxport.read(1) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 17:33:51.561690846 INFO receive_timing_info - Read() invoked 14438 us after write(), rxport.read(1) returned 0 bytes while blocked for 1 us. Rcv timeout.
2023-07-16 17:33:51.561692203 INFO receive_timing_info - Read() invoked 14440 us after write(), rxport.read(1) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 17:33:51.561694084 INFO receive_timing_info - Read() invoked 14441 us after write(), rxport.read(1) returned 0 bytes while blocked for 1 us. Rcv timeout.
2023-07-16 17:33:51.561695649 INFO receive_timing_info - Read() invoked 14443 us after write(), rxport.read(1) returned 0 bytes while blocked for 1 us. Rcv timeout.
```

... .. <intentionally omitted uninteresting read() log entries ... >
 <intentionally omitted uninteresting read() log entries ... >
 <intentionally omitted uninteresting read() log entries ... >

```

2023-07-16 17:33:52.561620759 INFO receive_timing_info -
2023-07-16 17:33:52.561622825 INFO receive_timing_info -
2023-07-16 17:33:52.561624658 INFO receive_timing_info -
2023-07-16 17:33:52.56162675 INFO receive_timing_info -
2023-07-16 17:33:52.561629146 INFO receive_timing_info -
2023-07-16 17:33:52.561631284 INFO receive_timing_info -
2023-07-16 17:33:52.561633332 INFO receive_timing_info -
2023-07-16 17:33:52.561635306 INFO receive_timing_info -
2023-07-16 17:33:52.561776316 INFO receive_timing_info -
2023-07-16 17:33:52.56177811 INFO receive_timing_info -
TRANSFER STALLED TIMEOUT ERROR: 'rxport::read()' repeatedly timed-out without receiving its requested incoming data. If not induced, inspect+verify the serial connections. Aborting.
Read() invoked 1014367 us after write(), rxport.read(1) returned 0 bytes while blocked for 1 us. Rcv timeout.
Read() invoked 1014370 us after write(), rxport.read(1) returned 0 bytes while blocked for 1 us. Rcv timeout.
Read() invoked 1014372 us after write(), rxport.read(1) returned 0 bytes while blocked for 1 us. Rcv timeout.
Read() invoked 1014373 us after write(), rxport.read(1) returned 0 bytes while blocked for 1 us. Rcv timeout.
Read() invoked 1014376 us after write(), rxport.read(1) returned 0 bytes while blocked for 1 us. Rcv timeout.
Read() invoked 1014378 us after write(), rxport.read(1) returned 0 bytes while blocked for 1 us. Rcv timeout.
Read() invoked 1014380 us after write(), rxport.read(1) returned 0 bytes while blocked for 1 us. Rcv timeout.
Read() invoked 1014382 us after write(), rxport.read(1) returned 0 bytes while blocked for 1 us. Rcv timeout.
Read() invoked 1014384 us after write(), rxport.read(1) returned 0 bytes while blocked for 1 us. Rcv timeout.

```

Proposed patch to v4.2.1 'serialport-rs' crate's 'set_timeout()' method for the next crate release. A simple patch- which improves the Windows platform read() behavior to eliminate the 0 timeout value read setting's known 'indefinite blocking' issue

This section discusses the Windows platform 'set_timeout()' trait method patch which eliminates the **read()** indefinite blocking when not all requested data is received. The patch conditionally alters (at run-time) the internal serial port timeout constants within the **set_timeout()** method body, located in the 'serialport-rs\src\windows\com.rs' source file. The patch revises the Windows **read()** timeout behavior for this scenario to return immediately from **read()** with any available received data at initial entry, or immediately with a timeout error when no data is available.

Below I show the current patch to the **set_timeout()** method, along with corresponding test run logs of the Windows **read()** behavior with a **set_timeout()** setting of 0, and where not all data is received for satisfying the posted **read()**'s buffer length. This log shows the **read()**'s indefinite blocking issue no longer occurs.

To be perfectly clear, the logs above were generated using the existing serialport-rs crate release v4.2.1, while the log below was generated running my patched version of the crate (per the indicated patch).

In the crate's current released Windows 'com.rs' file, at source line 242, is the **set_timeout()** method trait implementation shown here =>

```

fn set_timeout(&mut self, timeout: Duration) -> Result<()> {
    let milliseconds = timeout.as_secs() * 1000 + timeout.subsec_nanos() as u64 / 1_000_000;

    let mut timeouts = COMMTIMEOUTS {
        ReadIntervalTimeout: 0,
        ReadTotalTimeoutMultiplier: 0,
        ReadTotalTimeoutConstant: milliseconds as DWORD,
        WriteTotalTimeoutMultiplier: 0,
        WriteTotalTimeoutConstant: 0,
    };

    if unsafe { SetCommTimeouts(self.handle, &mut timeouts) } == 0 {
        return Err(super::error::last_os_error());
    }

    self.timeout = timeout;
}

```

```
    Ok(())
}
```

Below is the patched source file's replacement text - with a few explanatory comment lines =>

```
fn set_timeout(&mut self, timeout: Duration) -> Result<()> {
    let milliseconds = timeout.as_secs() * 1000 + timeout.subsec_nanos() as u64 / 1_000_000;
    let mut read_interval_timeout : u32 = 0u32; // Different internal setting value used if supplied time-out setting is 0 vs a non-zero input arg value.

    if milliseconds == 0 {
        read_interval_timeout = 0xFFFFFFFF; // The 'ReadIntervalTimeout' value setting of MAX_DWORD (0xFFFFFFFF), combined with a 0 'timeout duration parameter,
                                           // results in an immediate return from read() with whatever bytes are available at entry. If no data is available, read() immediately
                                           // returns a Timeout error (and no data). The 'ReadIntervalTimeout' value setting of 0, combined with a positive (> 0) 'timeout duration
                                           // parameter, waits for up to the indicated timeout period for the read()'s requested bytes to be available, returning
                                           // whatever is available at the end of the timeout duration or sooner if all requested data is received earlier.
                                           // If no data is available at the end of the timeout duration, then a timeout error is reported (with no data returned).
    }

    let mut timeouts = COMMTIMEOUTS {
        ReadIntervalTimeout: read_interval_timeout, // 0 or 0xFFFFFFFF,
        ReadTotalTimeoutMultiplier: 0 as DWORD, // milliseconds as DWORD,
        ReadTotalTimeoutConstant: milliseconds as DWORD,
        WriteTotalTimeoutMultiplier: 0,
        WriteTotalTimeoutConstant: 0,
    };

    if unsafe { SetCommTimeouts(self.handle, &mut timeouts) } == 0 {
        return Err(super::error::last_os_error());
    }
    self.timeout = timeout;
    Ok(())
}
```

While an improvement, unfortunately this simple patch doesn't improve or change the current Windows **read()** method's non-zero read timeout return responsiveness, where some data arrives prior to the time-out - but not the full requested amount. In this scenario the **read()** continues to block for the full timeout period, at which time it returns with the (lesser quantity) of available data. This is identical to the current Windows **read()** behavior in the same scenario, but can possibly be improved upon (see the next paragraph).

*Future work: Based on Microsoft's documentation concerning its native **Readfile()** System call, it seems it will require re-implementing entirely the **read()** trait method's implementation to utilize Microsoft's proprietary **ReadFile()** Overlapped IO capability, in hopes of improving this **read()** return behavior. This more sophisticated Windows **read()** method could potentially result in returning from the **read()** as soon as there is any received data – and before the entire read timeout period transpires. I haven't yet had an opportunity to (attempt to) develop and verify this Windows patch.*

I recognize what I'm describing here may be somewhat convoluted and unclear. If so, let me know and I'll attempt to better clarify.

Next is the initial section of the Windows log from executing the test application built with the simple patched crate, for the (previously) problematic Windows scenario of a **set_timeout()** value of 0 and a **read()** whose requested buffer size count of data never arrives. I've truncated the listing for brevity following receipt of the one and only byte, since with the patched crate our one second of logging (prior to a 'transfer stalled' timeout occurring) becomes quite large. Nevertheless, it's clear that the indefinite **read()** blocking problem doesn't occur with the patch =>

```
2023-07-16 19:26:01.4057889 INFO receive_timing_info - 'receive_timing_info' cross platform dual RS-232 port null modem cable connected rcv+xmt+timeout test and characterization tool: v1.0
2023-07-16 19:26:01.4061976 INFO receive_timing_info - Test setup: Platform='windows', Baud=200000, rx_tmo=0 ms, post_txdelays=0 ms, xfrstalledtmo=1000 ms, txlen=1, rxlen=2, repeat=1, fulldb=false
2023-07-16 19:26:01.4090968 INFO receive_timing_info - Test Logfile Name: 'D:\Users\ricej\windows_receive_timing_info.txt'
2023-07-16 19:26:01.456284 INFO receive_timing_info -
2023-07-16 19:26:01.4562918 INFO receive_timing_info - ** Start of cycle 1. **
2023-07-16 19:26:01.4562933 INFO receive_timing_info - Cycle 1 first phase -> Rx port = 'COM7', Tx port = 'COM6'.
2023-07-16 19:26:01.6590273 INFO receive_timing_info - txport.write() sent 1 bytes while blocked for 277 us. Read() invoked 9 us after write(), rxport.read(2) returned 0 bytes while blocked for 3 us. Rcv timeout.
2023-07-16 19:26:01.6590364 INFO receive_timing_info - Read() invoked 23 us after write(), rxport.read(2) returned 0 bytes while blocked for 1 us. Rcv timeout.
2023-07-16 19:26:01.6590396 INFO receive_timing_info - Read() invoked 26 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590417 INFO receive_timing_info - Read() invoked 29 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590457 INFO receive_timing_info - Read() invoked 31 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590478 INFO receive_timing_info - Read() invoked 35 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590516 INFO receive_timing_info - Read() invoked 37 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590534 INFO receive_timing_info - Read() invoked 41 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590552 INFO receive_timing_info - Read() invoked 43 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590572 INFO receive_timing_info - Read() invoked 45 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590595 INFO receive_timing_info - Read() invoked 47 us after write(), rxport.read(2) returned 0 bytes while blocked for 1 us. Rcv timeout.
2023-07-16 19:26:01.6590612 INFO receive_timing_info - Read() invoked 49 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590631 INFO receive_timing_info - Read() invoked 51 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590648 INFO receive_timing_info - Read() invoked 53 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590665 INFO receive_timing_info - Read() invoked 54 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590681 INFO receive_timing_info - Read() invoked 56 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590767 INFO receive_timing_info - Read() invoked 58 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590786 INFO receive_timing_info - Read() invoked 66 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590803 INFO receive_timing_info - Read() invoked 68 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.659083 INFO receive_timing_info - Read() invoked 70 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590901 INFO receive_timing_info - Read() invoked 73 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590922 INFO receive_timing_info - Read() invoked 80 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590941 INFO receive_timing_info - Read() invoked 82 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590957 INFO receive_timing_info - Read() invoked 84 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6590984 INFO receive_timing_info - Read() invoked 85 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591056 INFO receive_timing_info - Read() invoked 88 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591083 INFO receive_timing_info - Read() invoked 95 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591101 INFO receive_timing_info - Read() invoked 98 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591122 INFO receive_timing_info - Read() invoked 100 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591149 INFO receive_timing_info - Read() invoked 102 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.659117 INFO receive_timing_info - Read() invoked 105 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591186 INFO receive_timing_info - Read() invoked 107 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591202 INFO receive_timing_info - Read() invoked 108 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591239 INFO receive_timing_info - Read() invoked 110 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591291 INFO receive_timing_info - Read() invoked 114 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591313 INFO receive_timing_info - Read() invoked 119 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591333 INFO receive_timing_info - Read() invoked 121 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591416 INFO receive_timing_info - Read() invoked 123 us after write(), rxport.read(2) returned 0 bytes while blocked for 3 us. Rcv timeout.
2023-07-16 19:26:01.6591485 INFO receive_timing_info - Read() invoked 131 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591502 INFO receive_timing_info - Read() invoked 138 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591519 INFO receive_timing_info - Read() invoked 140 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591537 INFO receive_timing_info - Read() invoked 141 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591554 INFO receive_timing_info - Read() invoked 143 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591571 INFO receive_timing_info - Read() invoked 145 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
2023-07-16 19:26:01.6591589 INFO receive_timing_info - Read() invoked 147 us after write(), rxport.read(2) returned 0 bytes while blocked for 0 us. Rcv timeout.
```


[illegible][illegible]

[illegible][illegible]

[illegible]

19