[PATCH] io.c: avoid kwarg parsing in C API

07/07/2015 09:59 PM - normalperson (Eric Wong)

Status: Closed
Priority: Normal
Assignee: 
Target version: 

Description
rb_scan_args and hash lookups for kwargs in the C API are clumsy and slow. Instead of improving the C API for performance, use Ruby instead :)

Implement IO#read_nonblock and IO#write_nonblock in prelude.rb to avoid argument parsing via rb_scan_args and hash lookups.

This speeds up IO#write_nonblock and IO#read_nonblock benchmarks in both cases, including the original non-idtomatic case where the 'exception: false' hash is pre-allocated to avoid GC pressure.

Now, writing the kwargs in natural, idiomatic Ruby is fastest. I've added the noex2 benchmark to show this.

target 0: a (ruby 2.3.0dev (2015-07-08 trunk 51190) [x86_64-linux]) at "a/ruby"
target 1: b (ruby 2.3.0dev (2015-07-08 nonblock-kwarg 51190) [x86_64-linux]) at "b/ruby"

raw data:
[["io_nonblock_noex",
  [[2.5436805468052626, 2.5724728293716908, 2.4915440678596497],
  [2.478000810369849, 2.4285155069082975, 2.462410459294915]]],
 ["io_nonblock_noex2",
  [[3.012514788657427, 3.034533655270934, 2.9972082190215588],
  [2.135501991957426, 2.146781364455819, 2.0429874528199434]]]]

Elapsed time: 30.348340944 (sec)
-----------------------------------------------
benchmark results:
minimum results in each 3 measurements.
Execution time (sec)

<table>
<thead>
<tr>
<th>name</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_nonblock_noex</td>
<td>2.492</td>
<td>2.429</td>
</tr>
<tr>
<td>io_nonblock_noex2</td>
<td>2.997</td>
<td>2.043</td>
</tr>
</tbody>
</table>

Speedup ratio: compare with the result of 'a' (greater is better)
name b
<table>
<thead>
<tr>
<th>name</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_nonblock_noex</td>
<td>1.026</td>
<td></td>
</tr>
<tr>
<td>io_nonblock_noex2</td>
<td>1.467</td>
<td></td>
</tr>
</tbody>
</table>

Note: I plan to followup commits for other *_nonblock methods
Eventually, I even wish to deprecate rb_scan_args :D

For what it's worth, I'm more excited about this change than usual and hope to use prelude.rb more.

Associated revisions
Revision cee7691 - 11/12/2015 02:00 AM - normal
io.c: avoid kwarg parsing in C API

- benchmark/bm_io_nonblock_noex2.rb: new benchmark based on bm_io_nonblock_noex.rb
- io.c [io_read_nonblock]: move documentation to prelude.rb (io_write_nonblock): ditto (init_io): private, internal methods for prelude.rb use only
- prelude.rb [IO#read_nonblock]: wrapper + documentation (IO#write_nonblock): ditto [ruby-core:71439] [Feature #11339]
rb_scan_args and hash lookups for kwargs in the C API are clumsy and slow. Instead of improving the C API for performance, use Ruby instead :

Implement IO#read_nonblock and IO#write_nonblock in prelude.rb to avoid argument parsing via rb_scan_args and hash lookups.

This speeds up IO#write_nonblock and IO#read_nonblock benchmarks in both cases, including the original non-idiomatic case where the `exception: false` hash is pre-allocated to avoid GC pressure.

Now, writing the kwargs in natural, idiomatic Ruby is fastest. I've added the noex2 benchmark to show this.

2015-11-12 01:41:12 +0000
target 0: a (ruby 2.3.0dev (2015-11-11 trunk 52540) [x86_64-linux])

**target 1: b (ruby 2.3.0dev (2015-11-11 avoid-kwarg-capi 52540))**

benchmark results:
minimum results in each 10 measurements.
Execution time (sec)
<table>
<thead>
<tr>
<th>name</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_nonblock_noex</td>
<td>2.508</td>
<td>2.382</td>
</tr>
<tr>
<td>io_nonblock_noex2</td>
<td>2.950</td>
<td>1.882</td>
</tr>
</tbody>
</table>

Speedup ratio: compare with the result of `a` (greater is better)
<table>
<thead>
<tr>
<th>name</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_nonblock_noex</td>
<td>1.053</td>
</tr>
<tr>
<td>io_nonblock_noex2</td>
<td>1.567</td>
</tr>
</tbody>
</table>

Revision 52541 - 11/12/2015 02:00 AM - normalperson (Eric Wong)
io.c: avoid kwarg parsing in C API

- benchmark/bm_io_nonblock_noex2.rb: new benchmark based on bm_io_nonblock_noex.rb
- io.c (io_read_nonblock): move documentation to prelude.rb (io_write_nonblock): ditto (init_io): private, internal methods for prelude.rb use only
- prelude.rb (IO#read_nonblock): wrapper + documentation (IO#write_nonblock): ditto [ruby-core:71439] [Feature #11339]

rb_scan_args and hash lookups for kwargs in the C API are clumsy and slow. Instead of improving the C API for performance, use Ruby instead :

Implement IO#read_nonblock and IO#write_nonblock in prelude.rb to avoid argument parsing via rb_scan_args and hash lookups.

This speeds up IO#write_nonblock and IO#read_nonblock benchmarks in both cases, including the original non-idiomatic case where the `exception: false` hash is pre-allocated to avoid GC pressure.

Now, writing the kwargs in natural, idiomatic Ruby is fastest. I've added the noex2 benchmark to show this.

2015-11-12 01:41:12 +0000
target 0: a (ruby 2.3.0dev (2015-11-11 trunk 52540) [x86_64-linux])

**target 1: b (ruby 2.3.0dev (2015-11-11 avoid-kwarg-capi 52540))**

benchmark results:
minimum results in each 10 measurements.
Execution time (sec)
<table>
<thead>
<tr>
<th>name</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_nonblock_noex</td>
<td>2.508</td>
<td>2.382</td>
</tr>
<tr>
<td>io_nonblock_noex2</td>
<td>2.950</td>
<td>1.882</td>
</tr>
</tbody>
</table>

Speedup ratio: compare with the result of `a` (greater is better)
<table>
<thead>
<tr>
<th>name</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_nonblock_noex</td>
<td>1.053</td>
</tr>
<tr>
<td>io_nonblock_noex2</td>
<td>1.567</td>
</tr>
</tbody>
</table>

Revision 52541 - 11/12/2015 02:00 AM - normal
io.c: avoid kwarg parsing in C API

- benchmark/bm_io_nonblock_noex2.rb: new benchmark based on bm_io_nonblock_noex.rb
- io.c (io_read_nonblock): move documentation to prelude.rb (io_write_nonblock): ditto (Init_io): private, internal methods for prelude.rb use only
- prelude.rb (IO#read_nonblock): wrapper + documentation (IO#write_nonblock): ditto [ruby-core:71439] [Feature #11339]

rb_scan_args and hash lookups for kwargs in the C API are clumsy and slow. Instead of improving the C API for performance, use Ruby instead :) Implement IO#read_nonblock and IO#write_nonblock in prelude.rb to avoid argument parsing via rb_scan_args and hash lookups.

This speeds up IO#write_nonblock and IO#read_nonblock benchmarks in both cases, including the original non-idiomatic case where the 'exception: false' hash is pre-allocated to avoid GC pressure.

Now, writing the kwargs in natural, idiomatic Ruby is fastest. I've added the noex2 benchmark to show this.

2015-11-12 01:41:12 +0000
target 0: a (ruby 2.3.0dev 2015-11-11 trunk 52540) [x86_64-linux]

target 1: b (ruby 2.3.0dev 2015-11-11 avoid-kwarg-capi 52540)

benchmark results:
minimum results in each 10 measurements.
Execution time (sec)
name a b
io_nonblock_noex 2.508 2.382
io_nonblock_noex2 2.950 1.882

Speedup ratio: compare with the result of 'a' (greater is better)
name b
io_nonblock_noex 1.053
io_nonblock_noex2 1.567

Revision 52541 - 11/12/2015 02:00 AM - normal
io.c: avoid kwarg parsing in C API

- benchmark/bm_io_nonblock_noex2.rb: new benchmark based on bm_io_nonblock_noex.rb
- io.c (io_read_nonblock): move documentation to prelude.rb (io_write_nonblock): ditto (Init_io): private, internal methods for prelude.rb use only
- prelude.rb (IO#read_nonblock): wrapper + documentation (IO#write_nonblock): ditto [ruby-core:71439] [Feature #11339]

rb_scan_args and hash lookups for kwargs in the C API are clumsy and slow. Instead of improving the C API for performance, use Ruby instead :) Implement IO#read_nonblock and IO#write_nonblock in prelude.rb to avoid argument parsing via rb_scan_args and hash lookups.

This speeds up IO#write_nonblock and IO#read_nonblock benchmarks in both cases, including the original non-idiomatic case where the 'exception: false' hash is pre-allocated to avoid GC pressure.

Now, writing the kwargs in natural, idiomatic Ruby is fastest. I've added the noex2 benchmark to show this.

2015-11-12 01:41:12 +0000
target 0: a (ruby 2.3.0dev 2015-11-11 trunk 52540) [x86_64-linux]

target 1: b (ruby 2.3.0dev 2015-11-11 avoid-kwarg-capi 52540)

benchmark results:
minimum results in each 10 measurements.
Execution time (sec)
name a b
io_nonblock_noex 2.508 2.382
io_nonblock_noex2 2.950 1.882

Speedup ratio: compare with the result of 'a' (greater is better)
name b
io_nonblock_noex 1.053
io_nonblock_noex2  1.567

Revision 52541 - 11/12/2015 02:00 AM - normal
io.c: avoid kwarg parsing in C API

- benchmark/bm_io_nonblock_noex2.rb: new benchmark based on bm_io_nonblock_noex.rb
- io.c (io_read_nonblock): move documentation to prelude.rb (io_write_nonblock): ditto (init_io): private, internal methods for prelude.rb use only
- prelude.rb (IO#read_nonblock): wrapper + documentation (IO#write_nonblock): ditto [ruby-core:7439] [Feature #11339]

rb_scan_args and hash lookups for kwargs in the C API are clumsy and slow. Instead of improving the C API for performance, use Ruby instead:

Implement IO#read_nonblock and IO#write_nonblock in prelude.rb to avoid argument parsing via rb_scan_args and hash lookups.

This speeds up IO#write_nonblock and IO#read_nonblock benchmarks in both cases, including the original non-idiomatic case where the `exception: false` hash is pre-allocated to avoid GC pressure.

Now, writing the kwargs in natural, idiomatic Ruby is fastest. I've added the noex2 benchmark to show this.

2015-11-12 01:41:12 +0000
target 0: a (ruby 2.3.0dev (2015-11-11 trunk 52540) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-11 avoid-kwarg-capi 52540)

**target 1: b (ruby 2.3.0dev (2015-11-11 avoid-kwarg-capi 52540)**

benchmark results:
minimum results in each 10 measurements.
Execution time (sec)

<table>
<thead>
<tr>
<th>name</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_nonblock_noex</td>
<td>2.508</td>
<td>2.382</td>
</tr>
<tr>
<td>io_nonblock_noex2</td>
<td>2.950</td>
<td>1.882</td>
</tr>
</tbody>
</table>

Speedup ratio: compare with the result of `a` (greater is better)

<table>
<thead>
<tr>
<th>name</th>
<th>b</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>io_nonblock_noex</td>
<td>1.053</td>
<td></td>
</tr>
<tr>
<td>io_nonblock_noex2</td>
<td>1.567</td>
<td></td>
</tr>
</tbody>
</table>

Revision 52541 - 11/12/2015 02:00 AM - normal
io.c: avoid kwarg parsing in C API

- benchmark/bm_io_nonblock_noex2.rb: new benchmark based on bm_io_nonblock_noex.rb
- io.c (io_read_nonblock): move documentation to prelude.rb (io_write_nonblock): ditto (init_io): private, internal methods for prelude.rb use only
- prelude.rb (IO#read_nonblock): wrapper + documentation (IO#write_nonblock): ditto [ruby-core:7439] [Feature #11339]

rb_scan_args and hash lookups for kwargs in the C API are clumsy and slow. Instead of improving the C API for performance, use Ruby instead:

Implement IO#read_nonblock and IO#write_nonblock in prelude.rb to avoid argument parsing via rb_scan_args and hash lookups.

This speeds up IO#write_nonblock and IO#read_nonblock benchmarks in both cases, including the original non-idiomatic case where the `exception: false` hash is pre-allocated to avoid GC pressure.

Now, writing the kwargs in natural, idiomatic Ruby is fastest. I've added the noex2 benchmark to show this.

2015-11-12 01:41:12 +0000
target 0: a (ruby 2.3.0dev (2015-11-11 trunk 52540) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-11 avoid-kwarg-capi 52540)

**target 1: b (ruby 2.3.0dev (2015-11-11 avoid-kwarg-capi 52540)**

benchmark results:
minimum results in each 10 measurements.
Execution time (sec)

<table>
<thead>
<tr>
<th>name</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_nonblock_noex</td>
<td>2.508</td>
<td>2.382</td>
</tr>
<tr>
<td>io_nonblock_noex2</td>
<td>2.950</td>
<td>1.882</td>
</tr>
</tbody>
</table>
Speedup ratio: compare with the result of 'a' (greater is better)

<table>
<thead>
<tr>
<th>name</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_nonblock_noex</td>
<td>1.053</td>
</tr>
<tr>
<td>io_nonblock_noex2</td>
<td>1.567</td>
</tr>
</tbody>
</table>

Revision 528ff1b9 - 11/16/2015 11:25 PM - normal

socket: avoid arg parsing in rsock_s_recvfrom_nonblock

- ext/socket/init.c (rsock_s_recvfrom_nonblock): avoid arg parsing with C API [ruby-core:71439] [Feature #11339]
- ext/socket/basicsocket.c (bsock_recv_nonblock): adjust for above change, make private
- ext/socket/socket.c (sock_recvfrom_nonblock): ditto
- ext/socket/udpsocket.c (udp_recvfrom_nonblock): ditto
- ext/socket/lib/socket.rb (BasicSocket#recv_nonblock): new wrapper for private method, move RDoc (Socket#recvfrom_nonblock): ditto
  (UDPSocket#recvfrom_nonblock): ditto

Note, not adding bm_recv_nonblock.rb to benchmark/ directory
since it is non-portable. It is only in this commit message.

Benchmark results + code

```
target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52540) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52540) [x86_64-linux])
```

```
recv_nonblock

require 'socket'

nr = 1000000
msg = 'hello world'
buf = ''
size = msg.bytesize
UNIXSocket.pair(:SEQPACKET) do |a, b|
  nr.times do
    a.sendmsg(msg)
    b.recv_nonblock(size, 0, buf, exception: false)
  end
end
```

Raw data:

```
[["recv_nonblock",
  [1.83511221408844,
   1.8703329525887966,
   1.8448856547474861,
   1.859263762831688,
   1.8331583738327026],
  [1.5637447573244572,
   1.4062932096421719,
   1.4247371144592762,
   1.4108827747404575,
   1.4802536629140377]]
```

Elapsed time: 16.530452496 (sec)

Benchmark results:
minimum results in each 5 measurements.
Execution time (sec)

```
name a b
recv_nonblock 1.833 1.406
```

Speedup ratio: compare with the result of 'a' (greater is better)

```
name b
recv_nonblock 1.304
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52598 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 52598 - 11/16/2015 11:25 PM - normalperson (Eric Wong)

socket: avoid arg parsing in rsock_s_recvfrom_nonblock

- ext/socket/init.c (rsock_s_recvfrom_nonblock): avoid arg parsing with C API [ruby-core:71439] [Feature #11339]
- ext/socket/basicsocket.c (bsock_recv_nonblock): adjust for above change, make private
- ext/socket/socket.c (sock_recvfrom_nonblock): ditto
- ext/socket/udpsocket.c (udp_recvfrom_nonblock): ditto

09/16/2021
Note, not adding bm_recv_nonblock.rb to benchmark/ directory since it is non-portable. It is only in this commit message.

Benchmark results + code

target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52540) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52540) [x86_64-linux]

recv_nonblock

require 'socket'

nr = 1000000
msg = 'hello world'
buf = ''
size = msg.bytesize
UNIXSocket.pair(:SEQPACKET) do |a, b|
  nr.times do
    a.sendmsg(msg)
    b.recv_nonblock(size, 0, buf, exception: false)
  end
end

raw data:

[["recv_nonblock",
  [[1.83511221408844,
    1.8703329525887966,
    1.8448854747474861,
    1.859263762831688,
    1.8331583738327026],
   [1.5637447573244572,
    1.4062932096421719,
    1.4247371144592762,
    1.4108827747404575,
    1.4802536629140377]]]]

Elapsed time: **16.530452496 (sec)**

benchmark results:

minimum results in each 5 measurements.

Execution time (sec)

<table>
<thead>
<tr>
<th>name</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>recv_nonblock</td>
<td>1.833</td>
<td>1.406</td>
</tr>
</tbody>
</table>

Speedup ratio: compare with the result of 'a' (greater is better)

<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>recv_nonblock</td>
</tr>
</tbody>
</table>

Revision 52598 - 11/16/2015 11:25 PM - normal

socket: avoid arg parsing in rsock_s_recvfrom_nonblock

Note, not adding bm_recv_nonblock.rb to benchmark/ directory since it is non-portable. It is only in this commit message.

Benchmark results + code

target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52540) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52540) [x86_64-linux]
msg = 'hello world'
buf = 
size = msg.bytesize
UNIXSocket.pair(:SEQPACKET) do |a, b|
nr.times do
  a.sendmsg(msg)
b.recv_nonblock(size, 0, buf, exception: false)
end
end

raw data:
[["recv_nonblock",
  [[1.83511221408844,
    1.8703329525887966,
    1.844856547474861,
    1.849263762831688,
    1.8331583738327026],
   [1.5637447573244572,
    1.4062932096421719,
    1.4247371144592762,
    1.4108827747404575,
    1.4802536629140377]]]

Elapsed time: 16.530452496 (sec)
benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name          a       b
recv_nonblock  1.833   1.406

Speedup ratio: compare with the result of `a' (greater is better)
name          b
recv_nonblock  1.304

Revision 52598 - 11/16/2015 11:25 PM - normal
socket: avoid arg parsing in rsock_s_recvfrom_nonblock
  - ext/socket/init.c (sock_s_recvfrom_nonblock): avoid arg parsing with C API [ruby-core:71439] [Feature #11339]
  - ext/socket/basicsocket.c (sock_recv_nonblock): adjust for above change, make private
  - ext/socket/socket.c (sock_recvfrom_nonblock): ditto
  - ext/socket/udpsocket.c (udp_recvfrom_nonblock): ditto
  - ext/socket/lib/socket.rb (BasicSocket#recv_nonblock): new wrapper for private method, move RDoc (Socket#recvfrom_nonblock): ditto
  (UDPSocket#recvfrom_nonblock): ditto

Note, not adding bm_recv_nonblock.rb to benchmark/ directory since it is non-portable. It is only in this commit message.

Benchmark results + code
target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52540) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52540) [x86_64-linux])
Elapsed time: 16.530452496 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name          a       b
recv_nonblock   1.833   1.406

Speedup ratio: compare with the result of 'a' (greater is better)
name          b
recv_nonblock   1.304

Revision 52598 - 11/16/2015 11:25 PM - normal
socket: avoid arg parsing in rsock_s_recvfrom_nonblock

- ext/socket/init.c (rsock_s_recvfrom_nonblock): avoid arg parsing with C API [ruby-core:71439] [Feature #11339]
- ext/socket/basicsocket.c (bsock_recv_nonblock): adjust for above change, make private
- ext/socket/socket.c (sock_recvfrom_nonblock): ditto
- ext/socket/socket.c (udp_recvfrom_nonblock): ditto
- ext/socket/lib/socket.rb (BasicSocket#recv_nonblock): new wrapper for private method, move RDoc (Socket#recvfrom_nonblock): ditto
  (UDPSocket#recvfrom_nonblock): ditto

Note, not adding bm_recv_nonblock.rb to benchmark/ directory
since it is non-portable. It is only in this commit message.

Benchmark results + code
target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52540) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52540) [x86_64-linux])

recv_nonblock

require 'socket'

nr = 1000000
msg = 'hello world'
buf = ''
size = msg.bytesize
UNIXSocket.pair(:SEQPACKET) do |a, b|
  nr.times do
    a.sendmsg(msg)
    b.recv_nonblock(size, 0, buf, exception: false)
  end
end

raw data:

[["recv_nonblock",
  [1.83511221408844,
   1.8703329525887966,
   1.8448856547474861,
   1.859263762831688,
   1.8331583738327026],
  [1.5637447573244572,
   1.4062932096421719,
   1.4247371144592762,
   1.4108827747404575,
   1.4802536629140377]]

Elapsed time: 16.530452496 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name  a   b
recv_nonblock  1.833  1.406

Speedup ratio: compare with the result of `a' (greater is better)
name  b
recv_nonblock  1.304

Revision 52598 - 11/16/2015 11:25 PM - normal
socket: avoid arg parsing in rsock_s_recvfrom_nonblock

<table>
<thead>
<tr>
<th>name</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>recv_nonblock</td>
<td>1.833</td>
<td>1.406</td>
</tr>
</tbody>
</table>

Revision 52598 - 11/16/2015 11:25 PM - normal
socket: avoid arg parsing in rsock_s_recvfrom_nonblock

<table>
<thead>
<tr>
<th>name</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>recv_nonblock</td>
<td>1.304</td>
</tr>
</tbody>
</table>

Revision 52598 - 11/16/2015 11:25 PM - normal
socket: avoid arg parsing in rsock_s_recvfrom_nonblock

<table>
<thead>
<tr>
<th>name</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>recv_nonblock</td>
<td>1.833</td>
<td>1.406</td>
</tr>
</tbody>
</table>

Revision 52598 - 11/16/2015 11:25 PM - normal
socket: avoid arg parsing in rsock_s_recvfrom_nonblock

<table>
<thead>
<tr>
<th>name</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>recv_nonblock</td>
<td>1.304</td>
</tr>
</tbody>
</table>

Revision 52598 - 11/16/2015 11:25 PM - normal
socket: avoid arg parsing in rsock_s_recvfrom_nonblock

Note, not adding bm_recv_nonblock.rb to benchmark/ directory since it is non-portable. It is only in this commit message.

Benchmark results + code
target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52540) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52540) [x86_64-linux])

recv_nonblock

require 'socket'
rir = 1000000
msg = "hello world"
buf = 
size = msg.bytesize
UNIXSocket.pair(:SEQPACKET) do |a, b|
rir.times do
  a.sendmsg(msg)
b.recv_nonblock(size, 0, buf, exception: false)
end

raw data:

```
[["recv_nonblock",
  [[1.83511221408844,
    1.8703329525887966,
    1.8448856547474861,
    1.859263762831688,
    1.8331583738327026],
   [1.5637447573244572,
    1.4062932096421719,
    1.4247371144592762,
    1.4108827747404575,
    1.480253629140377]]]
```

Elapsed time: 16.530452496 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name  a   b
recv_nonblock  1.833  1.406

Speedup ratio: compare with the result of `a' (greater is better)
name  b
recv_nonblock  1.304

Revision bb6dfab2 - 11/16/2015 11:34 PM - normal
socket: Socket#connect_nonblock avoids arg parsing with C API

- ext/socket/init.c (sock_connect_nonblock): avoid arg parsing with C API [ruby-core:71439] [Feature #11339]
- ext/socket/basicsocket.c (sock_connect_nonblock): adjust for above change, make private
- ext/socket/socket.c (sock_connect_nonblock): ditto
- ext/socket/udpsocket.c (udp_connect_nonblock): ditto
- ext/socket/lib/socket.rb (BasicSocket#connect_nonblock): new wrapper for private method, move RDoc
  (Socket#recvfrom_nonblock): ditto
  (UDPSocket#recvfrom_nonblock): ditto

09/16/2021
connect_nonblock

require 'tempfile'
require 'socket'
require 'io/wait'
nr = 500000
Tempfile.create(%w(connect_nonblock .sock)) do |tmp|
  path = tmp.path
  File.unlink(path)
  s = UNIXServer.new(path)
  addr = Socket.sockaddr_un(path).freeze
  nr.times do
    c = Socket.new(Socket::AF_UNIX, Socket::SOCK_STREAM)
    while c.connect_nonblock(addr, exception: false) == :wait_writable
      c.wait_writable
    end
    s.accept.close
    c.close
  end
end

raw data:
[
  [
    "connect_nonblock",
    [4.014209181070328,
     3.847995534592227,
     3.981342639774084,
     4.471840236335993,
     3.7867715656757355],
    [3.639054525643587,
     3.58337214961648,
     3.525284394621849,
     3.5264607738533,
     3.511393066495657]
  ]
]

Elapsed time: 37.889623996 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name            a     b
connect_nonblock  3.787  3.511

Speedup ratio: compare with the result of ‘a’ (greater is better)
name            b
connect_nonblock  1.078

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52600 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 52600 - 11/16/2015 11:34 PM - normalperson (Eric Wong)
socket: Socket#connect_nonblock avoids arg parsing with C API

  * ext/socket/socket.c (sock_connect_nonblock): avoid argument parsing in C. [ruby-core:71439] [Feature #11339]
  * ext/socket/lib/socket.rb (Socket#connect_nonblock): new wrapper for private method, move RDoc

target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52540) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52540) [x86_64-linux])

connect_nonblock

require 'tempfile'
require 'socket'
require 'io/wait'
nr = 500000
Tempfile.create(%w(connect_nonblock .sock)) do |tmp|
  path = tmp.path
  File.unlink(path)
  s = UNIXServer.new(path)
end
addr = Socket.sockaddr_un(path).freeze
nr.times do
  c = Socket.new(Socket::AF_UNIX, Socket::SOCK_STREAM)
  while c.connect_nonblock(addr, exception: false) == :wait_writable
    c.wait_writable
  end
  s.accept.close
  c.close
end
end

raw data:

[["connect_nonblock",
[4.014209181070328, 3.8479955345392227, 3.981342639774084, 4.471840236335993, 3.7867715656757355],
[3.639054525643587, 3.58337214961648, 3.525284394621849, 3.52646067738533, 3.511393066495657]]]

Elapsed time: 37.889623996 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name     a       b
connect_nonblock   3.787   3.511

Speedup ratio: compare with the result of `a' (greater is better)
name     b
connect_nonblock   1.078

Revision 52600 - 11/16/2015 11:34 PM - normal
socket: Socket#connect_nonblock avoids arg parsing with C API

  ● ext/socket/socket.c (sock_connect_nonblock): avoid argument parsing in C. [ruby-core:71439] [Feature #11339]
  ● ext/socket/lib/socket.rb (Socket#connect_nonblock): new wrapper for private method, move RDoc

target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52540) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52540) [x86_64-linux]
Elapsed time: 37.889623996 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name a b
connect_nonblock 3.787 3.511

Speedup ratio: compare with the result of `a' (greater is better)
name b
connect_nonblock 1.078

Revision 52600 - 11/16/2015 11:34 PM - normal
socket: Socket#connect_nonblock avoids arg parsing with C API
- ext/socket/socket.c (sock_connect_nonblock): avoid argument parsing in C. [ruby-core:71439] [Feature #11339]
- ext/socket/lib/socket.rb (Socket#connect_nonblock): new wrapper for private method, move RDoc
target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52540) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52540) [x86_64-linux])

connect_nonblock
require 'tempfile'
require 'socket'
require 'io/wait'
nr = 500000
Tempfile.create(%w(connect_nonblock .sock)) do |tmp|
  path = tmp.path
  File.unlink(path)
  s = UNIXServer.new(path)
  addr = Socket.sockaddr_un(path).freeze
  nr.times do
    c = Socket.new(Socket::AF_UNIX, Socket::SOCK_STREAM)
    while c.connect_nonblock(addr, exception: false) == :wait_writable
      c.wait_writable
    end
    s.accept.close
    c.close
  end
end

raw data:
[["connect_nonblock",
  [[4.014209181070328,
    3.8479955345392227,
    3.981342639774084,
    4.471840236335993,
    3.7867715656757355],
    [3.639054525643587,
    3.58337214961648,
    3.52528396281849,
    3.52646067738533,
    3.511393066495657]]]]
name  a  b
connect_nonblock  3.787  3.511

Speedup ratio: compare with the result of `a' (greater is better)
name  b
connect_nonblock  1.078

Revision 52600 - 11/16/2015 11:34 PM - normal

socket: Socket#connect_nonblock avoids arg parsing with C API
- ext/socket/socket.c (sock_connect_nonblock): avoid argument parsing in C. [ruby-core:71439] [Feature #11339]
- ext/socket/lib/socket.rb (Socket#connect_nonblock): new wrapper for private method, move RDoc

target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52540) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52540) [x86_64-linux]

connect_nonblock

require 'tempfile'
require 'socket'
require 'io/wait'

nr = 500000
Tempfile.create(%w(connect_nonblock .sock)) do |tmp|
  path = tmp.path
  File.unlink(path)
  s = UNIXServer.new(path)
  addr = Socket.sockaddr_un(path).freeze
  nr.times do
    c = Socket.new(Socket::AF_UNIX, Socket::SOCK_STREAM)
    while c.connect_nonblock(addr, exception: false) == :wait_writable
      c.wait_writable
    end
    s.accept.close
    c.close
  end
end

raw data:

```
[['connect_nonblock",
  [4.014209181070328,
    3.8479955345392227,
    3.981342639774084,
    4.471840236335993,
    3.7867715656757355],
  [3.639054525643587,
    3.58337214961648,
    3.525284394621849,
    3.52646067738533,
    3.511393066495657]]
```

Elapsed time: 37.889623996 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name  a  b
connect_nonblock  3.787  3.511

Speedup ratio: compare with the result of `a' (greater is better)
name  b
connect_nonblock  1.078

Revision 52600 - 11/16/2015 11:34 PM - normal

socket: Socket#connect_nonblock avoids arg parsing with C API
- ext/socket/socket.c (sock_connect_nonblock): avoid argument parsing in C. [ruby-core:71439] [Feature #11339]
- ext/socket/lib/socket.rb (Socket#connect_nonblock): new wrapper for private method, move RDoc

target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52540) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52540) [x86_64-linux]
require 'tempfile'
require 'socket'
require 'io/wait'
nr = 500000
Tempfile.create(%w(connect_nonblock .sock)) do |tmp|
  path = tmp.path
  File.unlink(path)
  s = UNIXServer.new(path)
  addr = Socket.sockaddr_un(path).freeze
  nr.times do
    c = Socket.new(Socket::AF_UNIX, Socket::SOCK_STREAM)
    while c.connect_nonblock(addr, exception: false) == :wait_writable
      c.wait_writable
    end
    s.accept.close
    c.close
  end
end

raw data:

[["connect_nonblock",
[4.014209181070328,
3.8479955345392227,
3.981342639774084,
4.471840236335993,
3.7867715656757355],
[3.639054525643587,
3.58337214961648,
3.525284394621849,
3.52646067738533,
3.511393066495657]]

Elapsed time: 37.889623996 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name        a       b
connect_nonblock  3.787   3.511

Speedup ratio: compare with the result of `a` (greater is better)
name        b
connect_nonblock  1.078

Revision bee5b49a - 11/16/2015 11:40 PM - normal
socket: avoid arg parsing in rsock_s_accept_nonblock
- ext/socket/init.c (rsock_s_accept_nonblock): avoid parsing args [ruby-core:71439] [Feature #11339]
- ext/socket/rubysocket.h: adjust prototype
- ext/socket/socket.c (socket_accept_nonblock): make private
- ext/socket/tcpserver.c (tcp_accept_nonblock): ditto
- ext/socket/unixserver.c (unix_accept_nonblock): ditto
- ext/socket/lib/socket.rb (Socket#accept_nonblock): implement as wrapper, move RDoc (TCPserver#accept_nonblock): ditto
  (UNIXServer#accept_nonblock): ditto

target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52550) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52550) [x86_64-linux])

accept_nonblock

require 'tempfile'
require 'socket'
require 'io/wait'
nr = 500000
Tempfile.create(%w(accept_nonblock .sock)) do |tmp|
  path = tmp.path
  File.unlink(path)
  s = UNIXServer.new(path)
end
addr = Socket.sockaddr_un(path).freeze
nr.times do
  s.accept_nonblock(exception: false)
c = UNIXSocket.new(path)
s.wait_readable
  s.accept_nonblock(exception: false).close
  c.close
end
end

raw data:

```
["accept_nonblock",
[4.807877402752638, 4.930681671947241, 4.738454818725586, 4.69268161803484, 4.684675686061382],
[4.253904823213816, 4.255124930292368, 4.295955188572407, 4.248479191213846, 4.2133030295372368, 4.295955188572407, 4.248479191213846, 4.2133030295372368]]
```

Elapsed time: 45.123040065 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name a b
accept_nonblock 4.685 4.213

Speedup ratio: compare with the result of 'a' (greater is better)
name b
accept_nonblock 1.112

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52601 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 52601 - 11/16/2015 11:40 PM - normalperson (Eric Wong)
socket: avoid arg parsing in rsock_s_accept_nonblock

- ext/socket/init.c (rsock_s_accept_nonblock): avoid parsing args [ruby-core:71439] [Feature #11339]
- ext/socket/rubysocket.h: adjust prototype
- ext/socket/socket.c (sock_accept_nonblock): make private
- ext/socket/tcpserver.c (tcp_accept_nonblock): ditto
- ext/socket/unixserver.c (unix_accept_nonblock): ditto
- ext/socket/lib/socket.rb (Socket#accept_nonblock): implement as wrapper, move RDoc (TCPServer#accept_nonblock): ditto
  (UNIXServer#accept_nonblock): ditto

target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52550) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52550) [x86_64-linux]
Benchmark results: minimum results in each 5 measurements.
Execution time (sec)
name        a       b
accept_nonblock   4.685   4.213

Speedup ratio: compare with the result of 'a' (greater is better)
name        b
accept_nonblock   1.112

Revision 52601 - 11/16/2015 11:40 PM - normal
socket: avoid arg parsing in rsock_s_accept_nonblock

- ext/socket/init.c (rsock_s_accept_nonblock): avoid parsing args [ruby-core:71439] [Feature #11339]
- ext/socket/rubysocket.h: adjust prototype
- ext/socket/socket.c (socket_accept_nonblock): make private
- ext/socket/tcpserver.c (tcp_accept_nonblock): ditto
- ext/socket/unixserver.c (unix_accept_nonblock): ditto
- ext/socket/lib/socket.rb (Socket#accept_nonblock): implement as wrapper, move RDoc (TCPServer#accept_nonblock): ditto
- UNIXServer#accept_nonblock): ditto

Target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52550) [x86_64-linux])
Target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52550) [x86_64-linux])

accept_nonblock

```ruby
require 'tempfile'
require 'socket'
require 'io/wait'
nr = 500000
Tempfile.create(%w(accept_nonblock .sock)) do |tmp|
  path = tmp.path
  File.unlink(path)
  s = UNIXServer.new(path)
  addr = Socket.sockaddr_un(path).freeze
  nr.times do
    s.accept_nonblock(exception: false)
    c = UNIXSocket.new(path)
    s.wait_readable
    s.accept_nonblock(exception: false).close
    c.close
  end
end
```

raw data:

```ruby
["accept_nonblock",
[4.807877402752638, 4.930681671947241, 4.738454818725586, 4.69268161803484, 4.684675686061382],
[4.253904823213816, 4.255124930292368, 4.295955188572407, 4.248479191213846, 4.213303029537201]]
```
**Elapsed time: 45.123040065 (sec)**

Benchmark results:
Minimum results in each 5 measurements.
Execution time (sec)

<table>
<thead>
<tr>
<th>name</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept_nonblock</td>
<td>4.685</td>
<td>4.213</td>
</tr>
</tbody>
</table>

Speedup ratio: compare with the result of `a' (greater is better)

<table>
<thead>
<tr>
<th>name</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept_nonblock</td>
<td>1.112</td>
</tr>
</tbody>
</table>

Revision 52601 - 11/16/2015 11:40 PM - normal
Socket: avoid arg parsing in rsock_s_accept_nonblock

- ext/socket/init.c (rsock_s_accept_nonblock): avoid parsing args [ruby-core:71439] [Feature #11339]
- ext/socket/rubysocket.h: adjust prototype
- ext/socket/socket.c (sock_accept_nonblock): make private
- ext/socket/tcpserver.c (tcp_accept_nonblock): ditto
- ext/socket/unixserver.c (unix_accept_nonblock): ditto
- ext/socket/lib/socket.rb (Socket#accept_nonblock): implement as wrapper, move RDoc (TCPserver#accept_nonblock): ditto
- (UNIXServer#accept_nonblock): ditto

Target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52550) [x86_64-linux])
Target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52550) [x86_64-linux])

```ruby
accept_nonblock
require 'tempfile'
require 'socket'
require 'io/wait'
nr = 500000
Tempfile.create(%w(accept_nonblock .sock)) do |tmp|
  path = tmp.path
  # Tempfile.unlink(path)
  s = UNIXServer.new(path)
  addr = Socket.sockaddr_un(path).freeze
  nr.times do
    s.accept_nonblock(exception: false)
    c = UNIXSocket.new(path)
    s.wait_readable
    s.accept_nonblock(exception: false).close
    c.close
  end
end
```

Raw data:
```ruby
[["accept_nonblock",
  [4.80777402752638,
  4.930681671947241,
  4.738454818725586,
  4.69268161803484,
  4.684675686061382],
  [4.253904823213816,
  4.255124930292368,
  4.295955188572407,
  4.248479191213846,
  4.213303029537201]]
```
Speedup ratio: compare with the result of `a' (greater is better)

<table>
<thead>
<tr>
<th>name</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept_nonblock</td>
<td>1.112</td>
</tr>
</tbody>
</table>

Revision 52601 - 11/16/2015 11:40 PM - normal

socket: avoid arg parsing in rsock_s_accept_nonblock

- ext/socket/init.c (rsock_s_accept_nonblock): avoid parsing args [ruby-core:71439] [Feature #11339]
- ext/socket/rubysocket.h: adjust prototype
- ext/socket/socket.c (sock_accept_nonblock): make private
- ext/socket/tcpserver.c (tcp_accept_nonblock): ditto
- ext/socket/unixserver.c (unix_accept_nonblock): ditto
- ext/socket/lib/socket.rb (Socket#accept_nonblock): implement as wrapper, move RDoc (TCPServer#accept_nonblock): ditto
  (UNIXServer#accept_nonblock): ditto

target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52550) [x86_64-linux])

Revision 52601 - 11/16/2015 11:40 PM - normal

socket: avoid arg parsing in rsock_s_accept_nonblock

- ext/socket/init.c (rsock_s_accept_nonblock): avoid parsing args [ruby-core:71439] [Feature #11339]
- ext/socket/rubysocket.h: adjust prototype
- ext/socket/socket.c (sock_accept_nonblock): make private
- ext/socket/tcpserver.c (tcp_accept_nonblock): ditto

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)

<table>
<thead>
<tr>
<th>name</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept_nonblock</td>
<td>4.685</td>
<td>4.213</td>
</tr>
</tbody>
</table>

Speedup ratio: compare with the result of `a' (greater is better)

<table>
<thead>
<tr>
<th>name</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept_nonblock</td>
<td>1.112</td>
</tr>
</tbody>
</table>

Revision 52601 - 11/16/2015 11:40 PM - normal

socket: avoid arg parsing in rsock_s_accept_nonblock

- ext/socket/init.c (rsock_s_accept_nonblock): avoid parsing args [ruby-core:71439] [Feature #11339]
- ext/socket/rubysocket.h: adjust prototype
- ext/socket/socket.c (sock_accept_nonblock): make private
- ext/socket/tcpserver.c (tcp_accept_nonblock): ditto
```ruby
accept_nonblock

require 'tempfile'
require 'socket'
require 'io/wait'

nr = 500000
Tempfile.create(%w(accept_nonblock .sock)) do |tmp|
  path = tmp.path
  File.unlink(path)
  s = UNIXServer.new(path)
  addr = Socket.sockaddr_un(path).freeze
  nr.times do
    s.accept_nonblock(exception: false)
    c = UNIXSocket.new(path)
    s.wait_readable
    s.accept_nonblock(exception: false).close
    c.close
  end
end

raw data:

```

<table>
<thead>
<tr>
<th>name</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept_nonblock</td>
<td>4.685</td>
<td>4.213</td>
</tr>
</tbody>
</table>

Elapsed time: 45.123040065 (sec)

Benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name   a   b
accept_nonblock   4.685   4.213

Speedup ratio: compare with the result of `a' (greater is better)
name   b
accept_nonblock   1.112
```
buf = '.'
begin
r, w = UNIXSocket.pair(:SEQPACKET)
while i < nr
  i += 1
  w.sendmsg(msg)
  r.recvmsg_nonblock(1, exception: false)
end
ensure
r.close
w.close
end

raw data:

[["recvmsg_nonblock", [3.721657912940979, 3.6072621569037437, 3.580637402832508, 3.614185404032469, 3.6029579415917397], [2.4694008752703667, 2.4908322244882584, 2.505129844475746, 2.5037173740565777, 2.548359278589487]]]

Elapsed time: 30.646087052 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name     a       b
recvmsg_nonblock  3.581   2.469

Speedup ratio: compare with the result of ‘a’ (greater is better)
name     b
recvmsg_nonblock  1.450

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52602 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 52602 - 11/17/2015 12:58 AM - normalperson (Eric Wong)
socket (bsock_recvmsg_internal): avoid arg parsing

- ext/socket/andcdata.c (bsock_recvmsg_internal): avoid arg parsing (rsock_bsock_recvmsg): adjust for above change (rsock_bsock_recvmsg_nonblock): ditto [ruby-core:71439] [Feature #11339]
- ext/socket/rubysocketh: adjust prototypes for above
- ext/socket/basicsocket.c (rsock_init_basicsocket): adjust private methods
- ext/socket/lib/socket.rb (BasicSocket#recvmsg): wrapper method (BasicSocket#recvmsg_nonblock): ditto

target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52550) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52550) [x86_64-linux])

recvmsg_nonblock

require 'socket'
rnr = 1_000_000
i = 0
msg = '.'
buf = '.'
begin
  i += 1
  w.sendmsg(msg)
  r.recvmsg_nonblock(1, exception: false)
end
ensure
r.close
w.close
end
raw data:

```ruby
[["recvmsg_nonblock",
  [3.721687912940979,
   3.6072621569037437,
   3.580637402832508,
   3.614185404032469,
   3.6029579415917397],
  [2.4694008752703667,
   2.490832244882584,
   2.505127884475746,
   2.5037173740565777,
   2.548359278589487]]
```

Elapsed time: 30.646087052 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name    a       b
recvmsg_nonblock 3.581   2.469

Speedup ratio: compare with the result of a' (greater is better)
name    b
recvmsg_nonblock 1.450

Revision 52602 - 11/17/2015 12:58 AM - normal
socket (bsock_recvmsg_internal): avoid arg parsing
  - ext/socket/ancdata.c (bsock_recvmsg_internal): avoid arg parsing
  - ext/socket/rubysocket.h: adjust prototypes for above
  - ext/socket/basicsocket.c (rsock_init_basicsocket): adjust private methods
  - ext/socket/lib/socket.rb (BasicSocket#recvmsg): wrapper method
  - (rsock_bsock_recvmsg_nonblock): ditto

target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52550) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52550) [x86_64-linux])

```
raw data:
```

```
[["recvmsg_nonblock",
  [3.721687912940979,
   3.6072621569037437,
   3.580637402832508,
   3.614185404032469,
   3.6029579415917397],
  [2.4694008752703667,
   2.490832244882584,
   2.505127884475746,
   2.5037173740565777,
   2.548359278589487]]
```
Elapsed time: 30.646087052 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name a b
recvmsg_nonblock 3.581 2.469

Speedup ratio: compare with the result of `a' (greater is better)
name b
recvmsg_nonblock 1.450

Revision 52602 - 11/17/2015 12:58 AM - normal
socket (bsock_recvmsg_internal): avoid arg parsing

- ext/socket/ancdata.c (bsock_recvmsg_internal): avoid arg parsing (rsock_bsock_recvmsg): adjust for above change
  (rsock_bsock_recvmsg_nonblock): ditto [ruby-core:71439] [Feature #11339]
- ext/socket/rubysocket.h: adjust prototypes for above
- ext/socket/basicsocket.c (rsock_init_basicsocket): adjust private methods
- ext/socket/lib/socket.rb (BasicSocket#recvmsg): wrapper method (BasicSocket#recvmsg_nonblock): ditto

Revision 52602 - 11/17/2015 12:58 AM - normal
socket (bsock_recvmsg_internal): avoid arg parsing

- ext/socket/ancdata.c (bsock_recvmsg_internal): avoid arg parsing (rsock_bsock_recvmsg): adjust for above change
  (rsock_bsock_recvmsg_nonblock): ditto [ruby-core:71439] [Feature #11339]
- ext/socket/rubysocket.h: adjust prototypes for above
- ext/socket/basicsocket.c (rsock_init_basicsocket): adjust private methods
- ext/socket/lib/socket.rb (BasicSocket#recvmsg): wrapper method (BasicSocket#recvmsg_nonblock): ditto

target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52550) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52550) [x86_64-linux])
recvmsg_nonblock

```ruby
require 'socket
nr = 1_000_000
i = 0
msg = '.
buf = '.'
begin
  r, w = UNIXSocket.pair(:SEQPACKET)
  while i < nr
    i += 1
    w.sendmsg(msg)
    r.recvmsg_nonblock(1, exception: false)
  end
end
ensure
  r.close
  w.close
end

raw data:
[
  [
    ['recvmsg_nonblock',
      [3.721687912940979,
      3.6072621569037437,
      3.580637402832508,
      3.614185404032469,
      3.6029579415917397],
    [2.4694008752703667,
      2.4908322244882584,
      2.5051278844475746,
      2.5037173740565777,
      2.548359278589487]],
  ]
]

Elapsed time: 30.646087052 (sec)

Benchmark results:
- minimum results in each 5 measurements.
- Execution time (sec)
  - recvmsg_nonblock: 3.581 vs. 2.469 (Speedup ratio: 1.450)

Revision 52602 - 11/17/2015 12:58 AM - normal

recvmsg_nonblock

```
nr = 1_000_000
i = 0
msg = '.
buf = '.
begin
r, w = UNIXSocket.pair(:SEQPACKET)
while i < nr
i += 1
w.sendmsg(msg)
r.recvmsg_nonblock(1, exception: false)
end
ensure
r.close
w.close
end

raw data:
[["recvmsg_nonblock",
 [[3.721687912940979,
 3.6072621569037437,
 3.580637402832508,
 3.614185404032469,
 3.6029579415917397],
 [2.4694008752703667,
 2.4908322244882584,
 2.5051278844475746,
 2.5037173740565777,
 2.5483592785894874]]
]
Elapsed time: 30.646087052 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name a b
recvmsg_nonblock 3.581 2.469

Speedup ratio: compare with the result of 'a' (greater is better)
name a b
recvmsg_nonblock 1.450

Revision eda2441b - 11/17/2015 01:05 AM - normal
socket: avoid arg parsing in bsock_sendmsg_internal

sendmsg_nonblock
require 'socket'
nr = 1_000_000
i = 0
msg = '.
buf = '.
begin
r, w = UNIXSocket.pair(:SEQPACKET)
while i < nr
i += 1
w.sendmsg_nonblock(msg, exception: false)
r.recv(1, 0, buf)
end
ensure
r.close
w.close
end
raw data:

```
["sendmsg_nonblock",
[1.875997293740511,
1.8452614955604076,
1.8449317328631878,
1.8418389447033405,
1.869386937469244],
[1.5175109766423702,
1.4987873211503029,
1.498623799920082,
1.47918451577425,
1.5017359890043736]]
```

**Elapsed time: 16.775453245 (sec)**

**benchmark results:**
minimum results in each 5 measurements.
Execution time (sec)

<table>
<thead>
<tr>
<th>name</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>sendmsg_nonblock</td>
<td>1.842</td>
<td>1.479</td>
</tr>
</tbody>
</table>

Speedup ratio: compare with the result of 'a' (greater is better)

name       b
sendmsg_nonblock  1.245

**git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52603 b2dd03c8-39d4-4d8f-98ff-823fe69b080e**

**Revision 52603 - 11/17/2015 01:05 AM - normalperson (Eric Wong)**
socket: avoid arg parsing in bsock_sendmsg_internal

- `ext/socket/ancdata.c` (bsock_sendmsg_internal): avoid arg parsing [ruby-core:71439] Feature #11339: make private, adjust for above
- `ext/socket/rubysocket.h`: adjust prototypes (rsock_opt_false_p): remove
- `ext/socket/basicsocket.c` (rsock_init_basicsocket): define private methods
- `ext/socket/lib/socket.rb` (BasicSocket#sendmsg): new wrapper (BasicSocket#sendmsg_nonblock): ditto

target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52550) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52550) [x86_64-linux])

**sendmsg_nonblock**
```
require 'socket'
nr = 1_000_000
i = 0
msg = '.'
buf = '.
begin
  r, w = UNIXSocket.pair(:SEQPACKET)
  while i < nr
    i += 1
    w.sendmsg_nonblock(msg, exception: false)
    r.recv(1, 0, buf)
  end
  r.close
  w.close
end
```

raw data:

```
["sendmsg_nonblock",
[1.875997293740511,
1.8452614955604076,
1.8449317328631878,
1.8418389447033405,
1.869386937469244],
[1.5175109766423702,
1.4987873211503029,
1.498623799920082,
1.47918451577425,
1.5017359890043736]]
```
Elapsed time: 16.775453245 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name     a       b
sendmsg_nonblock  1.842   1.479

Speedup ratio: compare with the result of `a' (greater is better)
name     b
sendmsg_nonblock  1.245

Revision 52603 - 11/17/2015 01:05 AM - normal
socket: avoid arg parsing in bsock_sendmsg_internal

• ext/socket/ancdata.c (bsock_sendmsg_internal): avoid arg parsing [ruby-core:71439] Feature #11339: make private, adjust for above
  (rsock_bsock_sendmsg_nonblock): ditto
• ext/socket/rbysocket.h: adjust prototypes (rsock_opt_false_p): remove
• ext/socket/basicsocket.c (rsock_init_basicsocket): define private methods
• ext/socket/lib/socket.rb (BasicSocket#sendmsg): new wrapper (BasicSocket#sendmsg_nonblock): ditto

target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52550) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52550) [x86_64-linux])

sendmsg_nonblock

require 'socket'
nr = 1_000_000
i = 0
msg = '.'
buf = '.
begin
r, w = UNIXSocket.pair(:SEQPACKET)
while i < nr
  i += 1
  w.sendmsg_nonblock(msg, exception: false)
  r.recv(1, 0, buf)
end
ensure
  r.close
  w.close
end

raw data:
[
["sendmsg_nonblock",
[1.875997293740511,
1.8452614955604076,
1.8449317328631878,
1.8418389447033405,
1.869386937469244],
[1.5175109766423702,
1.4987873211503029,
1.4989623799920082,
1.47918451577425,
1.5017359890043736]]

Elapsed time: 16.775453245 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name     a       b
sendmsg_nonblock  1.842   1.479

Speedup ratio: compare with the result of `a' (greater is better)
name     b
sendmsg_nonblock  1.245
sendmsg_nonblock

require 'socket'
	nr = 1_000_000
	nr = 0

t = '
' buf = ''

begin
r, w = UNIXSocket.pair(:SEQPACKET)
while i < nr

w.sendmsg_nonblock(t, exception: false)

r.recv(1, 0, buf)

end

ensure
r.close
w.close

end

raw data:

[[
  "sendmsg_nonblock",
  [1.875997293740511,
   1.8452614955604076,
   1.8449317328631878,
   1.8418389447033405,
   1.869386937469244],
  [1.5175109766423702,
   1.4987873211503029,
   1.4989623799920082,
   1.47918451577425,
   1.5017359890043736]]

Elapsed time: 16.775453245 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name a b
sendmsg_nonblock 1.842 1.479

Speedup ratio: compare with the result of `a' (greater is better)
name b
sendmsg_nonblock 1.245

Revision 52603 - 11/17/2015 01:05 AM - normal
socket: avoid arg parsing in bsck_sendmsg_internal

  * ext/socket/ancdata.c (bsck_sendmsg internal): avoid arg parsing [ruby-core:71439] Feature #11339: make private, adjust for above
    (rsock sendmsg nonblock): ditto
  * ext/socket/rubysocket.h: adjust prototypes (rsock_opt_false_p): remove
  * ext/socket/basicsocket.c (rsock_init basicsocket): define private methods
  * ext/socket/lib/socket.rb (BasicSocket#sendmsg): new wrapper (BasicSocket#sendmsg_nonblock): ditto

target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52550) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52550) [x86_64-linux])

sendmsg_nonblock

09/16/2021
require 'socket'

nr = 1_000_000
i = 0
msg = '.'
buf = '.
begin
  r, w = UNIXSocket.pair(:SEQPACKET)
  while i < nr
    i += 1
    w.sendmsg_nonblock(msg, exception: false)
    r.recv(1, 0, buf)
  end
  r.close
  w.close
end

raw data:

[
  ["sendmsg_nonblock",
   [1.875997293740511,
    1.8452614955604076,
    1.844917328631878,
    1.841389447033405,
    1.869386937469244],
   [1.5175109766423702,
    1.4987873211503029,
    1.4989623799920082,
    1.47918451577425,
    1.5017359890043736]]
]

Elapsed time: 16.775453245 (sec)

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name             a       b
sendmsg_nonblock   1.842   1.479

Speedup ratio: compare with the result of `a' (greater is better)
name             b
sendmsg_nonblock   1.245

Revision 52603 - 11/17/2015 01:05 AM - normal
socket: avoid arg parsing in bsock_sendmsg_internal

  * ext/socket/ancdata.c (bsock_sendmsg_internal): avoid arg parsing [ruby-core:71439] Feature #11339: make private, adjust for above
  * ext/socket/bsock_sendmsg_nonblock): ditto
  * ext/socket/rubysocket.h: adjust prototypes (rsock_opt_false_p): remove
  * ext/socket/basicsocket.c (rsock_init_basicsocket): define private methods
  * ext/socket/lib/socket.rb (BasicSocket#sendmsg): new wrapper (BasicSocket#sendmsg_nonblock): ditto

  target 0: a (ruby 2.3.0dev (2015-11-12 trunk 52550) [x86_64-linux])
  target 1: b (ruby 2.3.0dev (2015-11-12 avoid-kwarg-capi 52550) [x86_64-linux])

sendmsg_nonblock

require 'socket'

nr = 1_000_000
i = 0
msg = '.'
buf = '.
begin
  r, w = UNIXSocket.pair(:SEQPACKET)
  while i < nr
    i += 1
    w.sendmsg_nonblock(msg, exception: false)
    r.recv(1, 0, buf)
  end
  r.close
  w.close
raw data:

```
[["sendmsg_nonblock",
[1.87997293740511, 1.8452614955604076, 1.8449317328631878, 1.8418389447033405, 1.869386937469244],
[1.517510976423702, 1.4987873211503029, 1.4989623799920082, 1.47918451577425, 1.5017359890043736]]
```

**Elapsed time: 16.775453245 (sec)**

benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name      a       b
sendmsg_nonblock  1.842   1.479

Speedup ratio: compare with the result of `a' (greater is better)
name      b
sendmsg_nonblock  1.245

**Revision b401937c - 12/07/2015 06:49 PM - normal**
doc/extension.rdoc: warn about kwargs performance in C [ci skip]

This existing API seems doomed performance-wise, and writing things in Ruby is nicer anyways. So discourage folks from using it.

[Feature #11339]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52925 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

**Revision 52925 - 12/07/2015 06:49 PM - normalperson (Eric Wong)**
doc/extension.rdoc: warn about kwargs performance in C [ci skip]

This existing API seems doomed performance-wise, and writing things in Ruby is nicer anyways. So discourage folks from using it.

[Feature #11339]

**Revision 52925 - 12/07/2015 06:49 PM - normal**
doc/extension.rdoc: warn about kwargs performance in C [ci skip]

This existing API seems doomed performance-wise, and writing things in Ruby is nicer anyways. So discourage folks from using it.

[Feature #11339]

**Revision 52925 - 12/07/2015 06:49 PM - normal**
doc/extension.rdoc: warn about kwargs performance in C [ci skip]

This existing API seems doomed performance-wise, and writing things in Ruby is nicer anyways. So discourage folks from using it.

[Feature #11339]

**Revision 52925 - 12/07/2015 06:49 PM - normal**
doc/extension.rdoc: warn about kwargs performance in C [ci skip]

This existing API seems doomed performance-wise, and writing things in Ruby is nicer anyways. So discourage folks from using it.

[Feature #11339]

**Revision 52925 - 12/07/2015 06:49 PM - normal**
doc/extension.rdoc: warn about kwargs performance in C [ci skip]

This existing API seems doomed performance-wise, and writing things in Ruby is nicer anyways. So discourage folks from using it.

[Feature #11339]
This existing API seems doomed performance-wise, and writing things in Ruby is nicer anyways. So discourage folks from using it.

[Feature #11339]

Revision 52925 - 12/07/2015 06:49 PM - normal
doc/extension.rdoc: warn about kwargs performance in C [ci skip]

This existing API seems doomed performance-wise, and writing things in Ruby is nicer anyways. So discourage folks from using it.

[Feature #11339]

History

#1 - 07/15/2015 07:48 PM - normalperson (Eric Wong)
normalperson@yhbt.net wrote:

Feature #11339: [PATCH] io.c: avoid kwarg parsing in C API
https://bugs.ruby-lang.org/issues/11339

Note: I plan to followup commits for other *_nonblock methods
Eventually, I even wish to deprecate rb_scan_args :D

For what it's worth, I'm more excited about this change than usual and hope to use prelude.rb more.

ko1/nobu/akr/others: any comments on this?

My main concern is increased parse time from prelude during startup; but we may translate prelude to iseq array and use rb_iseq_load, too. The parser seems to be the worst offender for startup performance nowadays.

#2 - 07/16/2015 03:48 AM - ko1 (Koichi Sasada)
On 2015/07/16 4:41, Eric Wong wrote:

normalperson@yhbt.net wrote:

Feature #11339: [PATCH] io.c: avoid kwarg parsing in C API
https://bugs.ruby-lang.org/issues/11339

Note: I plan to followup commits for other *_nonblock methods
Eventually, I even wish to deprecate rb_scan_args :D

For what it's worth, I'm more excited about this change than usual and hope to use prelude.rb more.

ko1/nobu/akr/others: any comments on this?

My main concern is increased parse time from prelude during startup; but we may translate prelude to iseq array and use rb_iseq_load, too. The parser seems to be the worst offender for startup performance nowadays.

We have some ideas to solve this issue. We discussed about solutions.

Known problems about C-methods parameters:
(P1) slow to parse kwargs with Hash
(P2) difficult to write scan_args
(P3) C-methods can't support Method#parameters

Solutions:

(1) Introduce wrapping Ruby methods into prelude.rb (your idea)
Pros. Easy to introduce.
Solves (P1-3).
Cons. Increase parse time at Ruby launch.
(2) Introduce new API to declare Ruby-like parameters for C-APis

like: `rb_define_method(klass, "xyzzy", klass_xyzzy, -1)`

(2-1)

```
-> rb_define_method?(klass, "xyzzy", klass_xyzzy,
"(m1, m2, o1=nil, o2=nil,
  `r, p1, p2, k1: 1, k2: 2")
```

```
VALUE klass_xyzzy(VALUE self, VALUE m1, VALUE m2, VALUE o1, VALUE o2,
  VALUE r, VALUE p1, VALUE p2, VALUE k1, VALUE k2)
```

or

(2-2)

```
-> rb_define_method?(klass, "xyzzy", klass_xyzzy,
  2 /* mandatory num 
  2 / optional num 
  1 / rest num 
  2 / post num 
  2 / kw num */,
  "m1", "m2",
  "o1", Qnil, "o2", Qnil,
  "r", "p1", "p2",
  "k1", Qnil, "k2", Qnil);
```

(2-3)

```
-> something = rb_define_method(klass, "xyzzy", klass_xyzzy, 9);
rb_define_method_argument(something, ...);
```

(or something like that)

Implementation: Make new `rb_iseq` only to call C func (klass_xyzzy, in this case). We have also need several issues.

Pros. Easy to specify parameters.
Solves (P1-3).
Cons. Difficult to design API (it should be compatible in future).
(2-1) introduces parse time at definition.

(3) Introduce new IDL (Interface Definition Language)

# File klass.??

class Klass
  def xyzzy(m1, m2, o1=nil, o2=nil, *r, p1, p2, k1: 1, k2: 2)
  # This decl. calls C func klass_xyzzy with parameters m1 to k2.
  # We can't write any code here.
  end
end

Translate klass.?? to something like (2).
We don't touch such APIs. No compatibility issues.

Pros. We don't need to design cool API.
Solves (P1-P3).
Cons. Need to design new language (IDL).

(4) Introduce new IDL like Ricsin

I made a system calls Ricsin, which enable to embed C code into Ruby code.

(sorry, written in Japanese)

# File klass.??

class Klass
  def xyzzy(m1, m2, o1=nil, o2=nil, r, p1, p2, k1: 1, k2: 2)
  # you can write any Ruby code here.
  C %Q{
  /Given string argument for C is C code. */
  klass_xyzzy(RV(m1), RV(m2), RV(o1), RV(o2)),
  ```
RV(r), RV(p1), RV(p2), RV(k1), RV(k2));
}
end
end

Compile this file into something C-extension.

Pros. Easy to write Extensions.
Easy (and efficient) to write exception handling code
without rb_protect(). rb_iterate() is same.
callback is difficult for C
Solves (P1-P3).
Cons. Allowing everything can make other issues.

Matz likes the middle of (3) and (4) (not allow everything, but allow
restricted). I like (4).

I'm okay to introduce (1) because it is easy and practical.
If we can make (2)-(4), then we can replace from (1) to new mechanism.

BTW, I'm working on making AOT compilation support (it will be continued
to (3) or (4)). Recent rb_iseq_t changes were for this purpose. So that
prelude.rb is nice benchmark for me.

Thanks,
Koichi

--
// SASADA Koichi at atdot dot net

#3 - 07/16/2015 07:08 AM - normalperson (Eric Wong)
SASADA Koichi ko1@atdot.net wrote:

On 2015/07/16 4:41, Eric Wong wrote:

normalperson@yhbt.net wrote:

Feature #11339: [PATCH] io.c: avoid kwarg parsing in C API
https://bugs.ruby-lang.org/issues/11339

Note: I plan to followup commits for other *_nonblock methods
Eventually, I even wish to deprecate rb_scan_args :D

For what it's worth, I'm more excited about this change than usual
and hope to use prelude.rb more.

ko1/nobu/akr/others: any comments on this?

My main concern is increased parse time from prelude during startup;
but we may translate prelude to iseq array and use rb_iseq_load, too.
The parser seems to be the worst offender for startup performance
nowadays.

We have some ideas to solve this issue. We discussed about solutions.

Known problems about C-methods parameters:
(P1) slow to parse kwargs with Hash
(P2) difficult to write scan_args
(P3) C-methods can't support Method#parameters

Thank you for response.

Solutions:

(1) Introduce wrapping Ruby methods into prelude.rb (your idea)
Pros. Easy to introduce.
Solves (P1-3).
Cons. Increase parse time at Ruby launch.
We cannot avoid parsing Ruby :) So I want to try to make parsing faster. Unfortunately, my parser knowledge is not much right now.

(2) Introduce new API to declare Ruby-like parameters for C-APIs

like: `rb_define_method(klass, "xyzzy", klass_xyzzy, -1)`

(2-1) `-> rb_define_method(??(klass, "xyzzy", klass_xyzzy, *(m1, m2, o1=nil, o2=nil, 
r, p1, p2, k1: 1, k2: 2))`)

OK, I had the same idea like this, too. But I do not want to introduce a new C API. IMHO, C API should be smaller, not bigger.

(3) Introduce new IDL (Interface Definition Language)

This may be OK... I don't see a big advantage over (1).

(4) Introduce new IDL like Ricsin

I made a system calls Ricsin, which enable to embed C code into Ruby code.

I think this is too ugly. One reason I like Ruby + (limited) C use is relatively good separation between the different languages.

Working on C-ext is mostly normal C, and not some weird in-between thing like Perl XS (gross!). Existing C programmers do not need to learn a lot of new things to work with current CRuby.

I think it is important that we can use C tools (gdb, ctags, sparse, etc...) can work without modification. But I still want to reduce C and use more Ruby[1].

I'm okay to introduce (1) because it is easy and practical.

OK, thank you. I will commit (1) this week and work on more prelude.rb for other IO/Socket kwargs methods.

If we can make (2)-(4), then we can replace from (1) to new mechanism.

BTW, I'm working on making AOT compilation support (it will be continued to (3) or (4)). Recent `rb_iseq_t` changes were for this purpose. So that `prelude.rb` is nice benchmark for me.

I want to speed up Ruby parsing + startup in general, too.

Along the lines of AOT: I also consider having something like ccache (self-managing size, only in $HOME, hashing-based) using `rb_iseq_load`.

I don't want to pollute users disk with too many compiled files; and it should use hashing so we may tweak formats/architectures and not worry about path conflicts with concurrently-installed Ruby versions.

We already have too many bug reports because C-exts/objs get shared.

[1] Fwiw, I like Rubinius philosophy a lot. However, the non-Free contribution platform and eventual implementation (slow startup time, "Ruby environment" vs being "another *nix tool" which CRuby/Perl are) put me off.

#4 - 07/16/2015 09:18 AM - ko1 (Koichi Sasada)

On 2015/07/16 16:01, Eric Wong wrote:

I'm okay to introduce (1) because it is easy and practical.

OK, thank you. I will commit (1) this week and work on more prelude.rb for other IO/Socket kwargs methods.
Ah, sorry. I think it is okay.
Matz, what do you think about?

--
// SASADA Koichi at atdot dot net

#5 - 11/10/2015 10:29 PM - normalperson (Eric Wong)
normalperson@yhbl.net wrote:

  https://bugs.ruby-lang.org/issues/11339

ko1 / matz: ping?

#6 - 11/11/2015 05:20 AM - matz (Yukihiro Matsumoto)
I am OK with the change. I don't really like something like __read_nonblock, but acceptable.

Matz.

#7 - 11/12/2015 02:01 AM - Anonymous
- Status changed from Open to Closed

Applied in changeset r52541.

io.c: avoid kwarg parsing in C API

- benchmark/bm_io_nonblock_noex2.rb: new benchmark based on bm_io_nonblock_noex.rb
- io.c (io_read_nonblock): move documentation to prelude.rb (io_write_nonblock): ditto (init_io): private, internal methods for prelude.rb use only
- prelude.rb (IO#read_nonblock): wrapper + documentation (IO#write_nonblock): ditto [ruby-core:71439] [Feature #11339]

rb_scan_args and hash lookups for kwargs in the C API are clumsy and slow. Instead of improving the C API for performance, use Ruby instead :)

Implement IO#read_nonblock and IO#write_nonblock in prelude.rb to avoid argument parsing via rb_scan_args and hash lookups.

This speeds up IO#write_nonblock and IO#read_nonblock benchmarks in both cases, including the original non-idiomatic case where the 'exception: false' hash is pre-allocated to avoid GC pressure.

Now, writing the kwargs in natural, idiomatic Ruby is fastest. I've added the noex2 benchmark to show this.

2015-11-12 01:41:12 +0000
target 0: a (ruby 2.3.0dev (2015-11-11 trunk 52540) [x86_64-linux])
target 1: b (ruby 2.3.0dev (2015-11-11 avoid-kwarg-capi 52540)

benchmark results:

<table>
<thead>
<tr>
<th>minimum results in each 10 measurements.</th>
<th>Execution time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>a</td>
</tr>
<tr>
<td>io_nonblock_noex</td>
<td>2.508</td>
</tr>
<tr>
<td>io_nonblock_noex2</td>
<td>2.950</td>
</tr>
</tbody>
</table>

Speedup ratio: compare with the result of 'a' (greater is better)

<table>
<thead>
<tr>
<th>minimum results in each 10 measurements.</th>
<th>Execution time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>b</td>
</tr>
<tr>
<td>io_nonblock_noex</td>
<td>1.053</td>
</tr>
<tr>
<td>io_nonblock_noex2</td>
<td>1.567</td>
</tr>
</tbody>
</table>

#8 - 11/12/2015 02:08 AM - normalperson (Eric Wong)

matz@ruby-lang.org wrote:

  I am OK with the change. I don't really like something like __read_nonblock, but acceptable.
OK, committed as r52541 for now. I don't have a better idea for hiding other than private + "__" prefix...

Will followup with other IO/Socket/SSL methods

#9 - 11/12/2015 09:58 AM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

    matz@ruby-lang.org wrote:

        I am OK with the change. I don't really like something like __read_nonblock, but acceptable.

        OK, committed as r52541 for now. I don't have a better idea for hiding other than private + "__" prefix...

        Will followup with other IO/Socket/SSL methods

Work-in-progress for rsock_s_recvfrom_nonblock-based methods. Likely to commit tomorrow...

Minor nit, this may reduce startup performance slightly more than prelude.rb changes because the RDoc comment stays in socket.rb instead of getting dropped at compile time:

http://80x24.org/spew/20151112-avoid-arg-parsing-in-rsock_s_recvfrom_nonblock@1/raw

I also noticed I forgot to update some RDoc for r50912; will adjust in a separate commit.

#10 - 11/13/2015 04:18 AM - normalperson (Eric Wong)

Entire series for sockets
http://80x24.org/spew/20151113041012.27235-1-e%4080x24.org/t.mbox.gz

ref: [ruby-core:71439] [Feature #11339]

benchmark/bm_accept_nonblock.rb  |  17 ++
benchmark/bm_connect_nonblock.rb |  18 ++
benchmark/bm_recvmsg_nonblock.rb |  16 ++
benchmark/bm_sendsmsg_nonblock.rb |  16 ++
ext/socket/ancdata.c            | 179 +++----------
ext/socket/basicsocket.c        |   73 +++--
ext/socket/init.c               |    23 +-+
ext/socket/lib/socket.rb        | 479 +++++++++++++++++++++++++++++++++++++++
ext/socket/rubysocket.c         |   26 +++
ext/socket/socket.c              | 203 +++++----
ext/socket/tcpserver.c          |   48 +++
ext/socket/udpsocket.c          |   63 ----
ext/socket/unixserver.c        |   47 ----
13 files changed, 659 insertions(+), 549 deletions(-)

Will commit soon

#11 - 11/13/2015 05:48 AM - ko1 (Koichi Sasada)

On 2015/11/13 13:18, Eric Wong wrote:

    benchmark/bm_accept_nonblock.rb  |  17 ++
    benchmark/bm_connect_nonblock.rb |  18 ++
    benchmark/bm_recvmsg_nonblock.rb |  16 ++
    benchmark/bm_sendmsg_nonblock.rb |  16 ++

could you consider to add some prefix like "bm_io" prefix?
We can understand what purpose is.

--
// SASADA Koichi at atdot dot net

#12 - 11/13/2015 07:08 AM - normalperson (Eric Wong)

09/16/2021 35/38
SASADA Koichi ko1@atdot.net wrote:

On 2015/11/13 13:18, Eric Wong wrote:

```
benchmark/bm_accept_nonblock.rb  |  17 ++
benchmark/bm_connect_nonblock.rb |  18 ++
benchmark/bm_recvmsg_nonblock.rb |  16 ++
benchmark/bm_sendmsg_nonblock.rb |  16 ++
```

could you consider to add some prefix like "bm_io" prefix?
We can understand what purpose is.

Ah, sorry, I was actually going to remove those before committing and only leave the code in the commit message. I don't want to cause portability problems for people running the suite.

#13 - 11/18/2015 02:08 AM - normalperson (Eric Wong)

Eric Wong normalperson@yhbt.net wrote:

Will followup with other IO/Socket/SSL methods

Done for normal socket, asked about SSL in [ruby-core:71538]

I might ignore optimizing ARGF.read_nonblock(... exception: false) for now since it (AFAIK) is not used frequently and the extra methods+ parsing time isn't worth it. 2.3 will be the first version to support "exception: false" on ARGF, even...

#14 - 12/02/2015 02:28 AM - normalperson (Eric Wong)

normalperson@yhbt.net wrote:

https://bugs.ruby-lang.org/issues/11339

For OpenSSL connect_nonblock/accept_nonblock, it seems to be not worth the effort for a 1% improvement given the overheads of various parts of OpenSSL. But I'm also not knowledgeable in OpenSSL, so my benchmark is also likely bogus:

http://80x24.org/spew/20151202020654.18328-1-e@80x24.org/

Additionally, for read_nonblock; calling read_nonblock/sysread via rb_funcall from inside osssl_ssl_read_internal would also require more work to avoid the hash allocation. Not sure if it's worth the effort at the moment.

#15 - 12/27/2015 10:49 PM - headius (Charles Nutter)

I don't usually jump in to grouse about CRuby changes, but this is really gross. We shouldn't be mucking up the core classes like this just to work around a missing optimization in the runtime.

I'd strongly recommend making a better rb_define_method system that allows you to pass keyword arguments directly to C code rather than having to make changes like this all over the place. Here's hoping that happens sooner rather than later and this change is only temporary.

We won't be making this change in JRuby, so our copy of socket.rb will diverge from MRIs. JRuby does not currently do allocation-free keyword arguments, but we will implement it soon for both Ruby targets and native targets.

#16 - 12/27/2015 11:00 PM - headius (Charles Nutter)

A suggestion for how to make kwarg-passing to C functions allocation-free: have a thread-local (or perhaps global, since CRuby doesn't run Ruby code in parallel) array you can populate with the key/value pairs coming out of the VM. Methods that want to receive opts directly can specify it through a new rb_define_method form, and the requirement is that they must process those keyword arguments before doing anything else, so that global store can be re-used.

That is just a quick and dirty way to do it, but it would eliminate the need for hacks like this and make kwarg passing to C functions nearly free, as it is from Ruby to Ruby.

#17 - 12/27/2015 11:48 PM - normalperson (Eric Wong)

headius@headius.com wrote:
A suggestion for how to make kwarg-passing to C functions allocation-free: have a thread-local (or perhaps global, since CRuby doesn't run Ruby code in parallel) array you can populate with the key/value pairs coming out of the VM. Methods that want to receive opts directly can specify it through a new rb_define_method form, and the requirement is that they must process those keyword arguments before doing anything else, so that global store can be re-used.

Using globals or TLS would introduce subtle reentrancy problems when calls are nested. I don't want to create more C-APIs we need to support long-term, either.

Likely we will introduce something like prelude.rb into the extension build system (and/or use the new iseq loader features). It would speed up load times, too.

JRuby doesn't use our current prelude.rb, I hope.

Fwiw, I like the Rubinius philosophy of using Ruby as much as possible a lot and tried to contribute there back in the day.

I just do not like their non-Free-service-based development, C++, the isolated "Ruby environment" model, or slow startup times.

Unsubscribe: ruby-core-request@ruby-lang.org?subject=unsubscribe
http://lists.ruby-lang.org/cgi-bin/mailman/options/ruby-core

#18 - 12/28/2015 12:13 AM - headius (Charles Nutter)

Eric Wong wrote:

Using globals or TLS would introduce subtle reentrancy problems when calls are nested. I don't want to create more C-APIs we need to support long-term, either.

The global would only be used between the runtime making a call with keyword arguments and the consumption of those arguments for variable initialization in a C method. There's no nesting of calls, and once the C method has consumed the values in the global (i.e. immediately before doing anything else), it is not touched until the next call with keywords from Ruby to C.

JRuby doesn't use our current prelude.rb, I hope.

We use parts of it, to stay in alignment with the load order of things like RubyGems and did_you_mean. We won't use the IO parts.

The socket changes are in socket.rb, which is part of stdlib and we do use it. I'll have to maintain another patch to remove this stuff.

Fwiw, I like the Rubinius philosophy of using Ruby as much as possible a lot and tried to contribute there back in the day.

Yes, except that this doesn't buy anything because 99% of the logic of these methods still lives within C code. If you were also moving the body of these nonblock methods into Ruby, I'd see some value. You've removed a couple lines of C and added a couple lines of Ruby. And now there's going to be an extra method in stack traces and an extra Ruby frame allocated for every call.

#19 - 12/28/2015 12:22 AM - headius (Charles Nutter)

Comparing stack traces:

```ruby
[] ~/projects/ruby $ ruby23 -rsocket -e "t = TCPSocket.new('google.com', 80); t.read_nonblock(1)"
<internal:prelude>:76:in `__read_nonblock': Resource temporarily unavailable - read would block (IO::EAGAINWaitReadable)
  from <internal:prelude>:76:in `read_nonblock'
  from -e:1:in `<main>'

[] ~/projects/ruby $ ruby22 -rsocket -e "t = TCPSocket.new('google.com', 80); t.read_nonblock(1)"
-e:1:in `read_nonblock': Resource temporarily unavailable - read would block (IO::EAGAINWaitReadable)
  from -e:1:in `<main>'
```

#20 - 12/28/2015 01:18 AM - normalperson (Eric Wong)

headius@headius.com wrote:
Eric Wong wrote:

Using globals or TLS would introduce subtle reentrancy problems when calls are nested. I don’t want to create more C-APIs we need to support long-term, either.

The global would only be used between the runtime making a call with keyword arguments and the consumption of those arguments for variable initialization in a C method. There’s no nesting of calls, and once the C method has consumed the values in the global (i.e. immediately before doing anything else), it is not touched until the next call with keywords from Ruby to C.

The current rb_get_kwargs() calls may be delayed until the keyword is actually needed (to avoid unnecessary hash lookups).

Anyways, if somebody can design a good API for internal use, we can use it. Current rb_get_kwargs() and even rb_scan_args() are inefficient and even error-prone:

http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/68507

Fwiw, I like the Rubinius philosophy of using Ruby as much as possible a lot and tried to contribute there back in the day.

Yes, except that this doesn’t buy anything because 99% of the logic of these methods still lives within C code. If you were also moving the body of these nonblock methods into Ruby, I’d see some value. You’ve removed a couple lines of C and added a couple lines of Ruby. And now there’s going to be an extra method in stack traces and an extra Ruby frame allocated for every call.

It’s still a work-in-progress, obviously; but the current state already speeds things up for ‘exception: false’ users. Implementation details will change, of course; but we need freedom to change them by having a smaller public C API.

Unsubscribe:
ruby-core-request@ruby-lang.org?subject=unsubscribe
http://lists.ruby-lang.org/cgi-bin/mailman/options/ruby-core

#21 - 12/28/2015 01:50 AM - headius (Charles Nutter)

Eric Wong wrote:

Anyways, if somebody can design a good API for internal use, we can use it. Current rb_get_kwargs() and even rb_scan_args() are inefficient and even error-prone:

http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/68507

Something along the lines of vm_get_current_kwarg(name) could simply use existing interpreter structures for managing in-flight keyword arguments.

It’s still a work-in-progress, obviously; but the current state already speeds things up for ‘exception: false’ users. Implementation details will change, of course; but we need freedom to change them by having a smaller public C API.

I just hope this isn’t a trend, since it’s a pretty ugly way to work around keyword arguments not optimizing well for calls from Ruby to C. Imagine a world where every core method that accepts keywords went the same route. I think we want a better option before that happens.

Files

0001-io.c-avoid-kwarg-parsing-in-C-API.patch  6.88 KB  07/07/2015  normalperson (Eric Wong)