Ruby - Feature #8658

Process.clock_gettime

07/19/2013 09:32 PM - akr (Akira Tanaka)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

How about adding a new method, Process.clock_gettime(clk_id) ?

Recently there were two feature request for measuring time.
Feature #8640 https://bugs.ruby-lang.org/issues/8640
Feature #8096 https://bugs.ruby-lang.org/issues/8096

It seems they are somewhat different.

clock_gettime() function defined by POSIX is a good
candidate for providing as a method.
I think it can supports the both request.

Also, it has less possible design choices than the requests
because clock_gettime() is defined by POSIX.
People familiar to POSIX can learn the method more easily.

I wrote a patch to implement Process.clock_gettime.
This method can be used as follows.

```
% ./ruby -e 'p Process.clock_gettime(Process::CLOCK_MONOTONIC)'
2701692957811563
```

Several considerations:

I implemented the method as a module function of Process.
It is same as Process.times.
I expect clock_gettime is used mainly for measuring
time interval and wall clock time is not important.
So I didn't use Time.

The method returns a number of nanoseconds as an integer.
It is not so unexpected if user knows clock_gettime() in POSIX.

clock_gettime() returns it as struct timespec
which contains two fields: tv_sec and tv_nsec.

Although tv_sec is time_t, Time is not appropriate because
the origin (zero) can be other than the Epoch.
Actually CLOCK_MONOTONIC means elapsed time since
the system start-up time on Linux.

Also, I expect the result is subtracted in most case:

```
t1 = Process.clock_gettime(...)
...
t2 = Process.clock_gettime(...)
t = t2 - t1
```

So the result should be easy to subtract.
An array such as [sec, nsec] is difficult to subtract.

The result is an integer, not a float.
IEEE 754 double is not enough to represent the result
of clock_gettime(CLOCK_REALTIME).
It contains 19 digits in decimal now but IEEE 754 double
can represent only 15 digits.

On LP64 systems, Fixnum can represent 2**62-1.**
So (262-1)/(365.25*24*60*60*1e9)=146.1 years are representable
without object allocation.

On ILP32 and LLP64 systems, Fixnum can represent 2**30-1.**
So (230-1)/1e9=1.07 seconds are representable
without object allocation.
This means Bignum allocations are mostly required except
the origin is very recent.

clock_gettime() is defined by POSIX.
Linux, NetBSD, FreeBSD, OpenBSD has it, at least.

If clock_gettime() is not available,
an emulation layer for CLOCK_REALTIME is implementable
using gettimeofday().
(not implemented yet, though.)

Any comments?

Related issues:

Related to Ruby - Feature #8096: introduce Time.current_timestamp	**Feedback**
Related to Ruby - Feature #8640: Add Time#elapsed to return nanoseconds since...	**Open**
Related to Ruby - Feature #8777: Process.mach_absolute_time	**Closed**

Associated revisions

Revision b26f8003c301e7cfbeb5b9329f3254e858222cdc - 08/15/2013 05:18 PM - naruse (Yui NARUSE)

- process.c (rb_clock_gettime): add CLOCK_MONOTONIC support on OS X.
  http://developer.apple.com/library/mac/qa/qa1398/_index.html
  [Feature #8658]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@42573 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision b26f8003 - 08/15/2013 05:18 PM - naruse (Yui NARUSE)

- process.c (rb_clock_gettime): add CLOCK_MONOTONIC support on OS X.
  http://developer.apple.com/library/mac/qa/qa1398/_index.html
  [Feature #8658]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@42573 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 07/20/2013 04:52 AM - kosaki (Motohiro KOSAKI)

First, Process.times() returns user time and system time and they are process specific. But Process::CLOCK_MONOTONIC is not per-process time.

Second, Linux's CLOCK_MONOTONIC_RAW has the same behavior BSD's CLOCK_MONOTONIC. And, an application which measures a performance need to use CLOCK_MONOTONIC_RAW for avoiding ntp confusing. Then, we should do 1) exporse CLOCK_MONOTONIC_RAW or 2) Process.clock_gettime(Process::CLOCK_MONOTONIC) uses CLOCK_MONOTONIC_RAW internally.

Third, using float is a good ruby convention. If we need to use inter (for precision and performance?), the method should have a precision explanation, likes get_time_nanosecond. I mean, ruby interpreter can't warn nor detect following mistake.

a = foo # this is usec
b = bar # this is nsec
c = a + b

then, we should warn by method name verbosely. IMHO.

#2 - 07/20/2013 07:39 PM - akr (Akira Tanaka)

- File clock_gettime-2.patch added

kosaki (Motohiro KOSAKI) wrote:

> First, Process.times() returns user time and system time and they are process specific. But Process::CLOCK_MONOTONIC is not per-process time.

Yes. Users can choose any clock with Process.clock_gettime unlike other proposals ([#8640](#), [#8096](#)).

It seems many people use CLOCK_REALTIME to measure a time interval, though.

> Second, Linux's CLOCK_MONOTONIC_RAW has the same behavior BSD's CLOCK_MONOTONIC. And, an application which measures a performance need to use CLOCK_MONOTONIC_RAW for avoiding ntp confusing. Then, we should do 1) exporse CLOCK_MONOTONIC_RAW or 2) Process.clock_gettime(Process::CLOCK_MONOTONIC) uses CLOCK_MONOTONIC_RAW internally.

OS specific CLOCK_* constants can be defined.
Since Process.clock_gettime is a primitive, exchange clk_id is not a good idea.

> Third, using float is a good ruby convention. If we need to use inter (for precision and performance?), the method should have a precision explanation, likes get_time_nanosecond. I mean, ruby interpreter can't warn nor detect following mistake.
>
> a = foo # this is usec
> b = bar # this is nsec
> c = a + b
>
> then, we should warn by method name verbosely. IMHO.

Hm. It is acceptable as far as the exact result (number of nanoseconds) can be obtained.

After thinking while, I find Process.clock_gettime(clk_id, unit).
unit is an optional argument and :nanoseconds specifies the nanoseconds.
This can help performance on ILP33 because :microseconds with CLOCK_PROCESS_CPUTIME_ID
will not use Bignum until 1073 seconds after process start up.

I updated the patch.

#3 - 07/21/2013 02:53 PM - duerst (Martin Dürst)

Hello Akira,

On 2013/07/19 21:32, akr (Akira Tanaka) wrote:

> On LP64 systems, Fixnum can represent 2**62-1.**
> **So (2**62-1)/(365.25*24*60*60*1e9)=146.1 years are representable
> without object allocation.
>
> On ILP32 and LLP64 systems, Fixnum can represent 2**30-1.**
> **So (2**30-1)/1e9=1.07 seconds are representable
> without object allocation.
> This means Bignum allocations are mostly required except
> the origin is very recent.

Don't your calculations ignore the fact that Fixnums are signed? Or do
you have a way to use negative amounts of nanoseconds, too?

Regards,   Martin.

#4 - 07/21/2013 04:59 PM - akr (Akira Tanaka)

2013/7/21 "Martin J. Dürst" duerst@it.aoyama.ac.jp:

> On 2013/07/19 21:32, akr (Akira Tanaka) wrote:
>
>> On LP64 systems, Fixnum can represent 2**62-1.**
>> **So (2**62-1)/(365.25*24*60*60*1e9)=146.1 years are representable
>> without object allocation.
>>
>> On ILP32 and LLP64 systems, Fixnum can represent 2**30-1.**
>> **So (2**30-1)/1e9=1.07 seconds are representable
>> without object allocation.
>> This means Bignum allocations are mostly required except
>> the origin is very recent.
>
> Don't your calculations ignore the fact that Fixnums are signed? Or do you
> have a way to use negative amounts of nanoseconds, too?

% ./ruby -ve 'p (2**62-1).class, (2**62).class'
ruby 2.1.0dev (2013-07-21 trunk 42095) [x86_64-linux]
Fixnum
Bignum

% ./ruby -ve 'p (2**30-1).class, (2**30).class'
ruby 2.1.0dev (2013-07-21 trunk 42095) [i686-linux]
Fixnum
Bignum

What's the problem?

Tanaka Akira

#5 - 07/21/2013 07:15 PM - Eregon (Benoit Daloze)

While I appreciate Ruby is not always taking the lowest common denominator for functionality (fork, etc),
we need a counterpart for Windows and OS X at least.

https://github.com/copiousfreetime/hitimes does it pretty nicely and I would really enjoy seeing the functionality to measure a precise interval in core.

We should have a unified method for clock_gettime(), mach_absolute_time() and QueryPerformanceCounter().
Actually, I think it would be better to have an API allowing to have the time difference directly in seconds (higher level and the concern for allocating is no more relevant, any unit can be used internally).

And this feature is not always compatible with the timestamp (given they are monotonic clocks), so I think there should be two separate methods.

naruse gave a very useful link in #8096, http://www.python.org/dev/peps/pep-0418/ .
I do not wish for a so large API, but I think we should have the timestamp functionality like time.time()
and a precise performance counter like time.perf_counter().

I would be fine having the clock_id as a parameter for supporting platforms if it proves really useful.

#6 - 07/24/2013 12:23 AM - akr (Akira Tanaka)

2013/7/21 Eregon (Benoit Daloze) redmine@ruby-lang.org:

> Issue #8658 has been updated by Eregon (Benoit Daloze).
>
> While I appreciate Ruby is not always taking the lowest common denominator for functionality (fork, etc),
> we need a counterpart for Windows and OS X at least.

Users of such OSs can contribute an emulation function for clock_gettime.

> naruse gave a very useful link in #8096, http://www.python.org/dev/peps/pep-0418/ .
> I do not wish for a so large API, but I think we should have the timestamp functionality like time.time()
> and a precise performance counter like time.perf_counter().

PEP 0418 mentions that python provide clock_gettime as time.clock_gettime.
PEP 0418 doesn't mean providing clock_gettime itself is bad idea.

Higer level methods may be useful but what I intend in this issue is a low level primitive.

Tanaka Akira

#7 - 07/24/2013 01:22 AM - Eregon (Benoit Daloze)

akr (Akira Tanaka) wrote:

> > While I appreciate Ruby is not always taking the lowest common denominator for functionality (fork, etc),
> > we need a counterpart for Windows and OS X at least.
>
> Users of such OSs can contribute an emulation function for clock_gettime.

A very poor one as mapping to Linux/UNIX constants would just confuse people.
I do not think the UNIX API clock_gettime() for this is the most suitable,
it does not abstract the functionality and the name/usage is not very ruby-like.

I think FFI would be a good way if someone need direct access to that low-level C function (except for accessing the constants, that would not be handy).

> > naruse gave a very useful link in #8096, http://www.python.org/dev/peps/pep-0418/ .
> > I do not wish for a so large API, but I think we should have the timestamp functionality like time.time()
> > and a precise performance counter like time.perf_counter().
>
> PEP 0418 mentions that python provide clock_gettime as time.clock_gettime.
> PEP 0418 doesn't mean providing clock_gettime itself is bad idea.

I believe providing a method which is only available in a quite restricted set of platforms is to be avoided.
In Python it is simply not defined on non-supporting platforms.

> Higer level methods may be useful but what I intend in this issue is a
> low level primitive.

To which use-cases other than benchmarking do you think?

I want Ruby to propose a nice and precise way to benchmark code *not* requiring the user to know about every detail of available clocks/timers under every platform.

#8 - 07/24/2013 03:14 AM - jonforums (Jon Forums)

Eregon (Benoit Daloze) wrote:

> akr (Akira Tanaka) wrote:
>
> > > While I appreciate Ruby is not always taking the lowest common denominator for functionality (fork, etc),
> > > we need a counterpart for Windows and OS X at least.
> >
> > Users of such OSs can contribute an emulation function for clock_gettime.
>
> A very poor one as mapping to Linux/UNIX constants would just confuse people.
> I do not think the UNIX API clock_gettime() for this is the most suitable,
> it does not abstract the functionality and the name/usage is not very ruby-like.
>
> I think FFI would be a good way if someone need direct access to that low-level C function (except for accessing the constants, that would not be handy).
>
> > > naruse gave a very useful link in #8096, http://www.python.org/dev/peps/pep-0418/ .
> > > I do not wish for a so large API, but I think we should have the timestamp functionality like time.time()
> > > and a precise performance counter like time.perf_counter().
> >
> > PEP 0418 mentions that python provide clock_gettime as time.clock_gettime.
> > PEP 0418 doesn't mean providing clock_gettime itself is bad idea.
>
> I believe providing a method which is only available in a quite restricted set of platforms is to be avoided.
> In Python it is simply not defined on non-supporting platforms.

It's great to see a focus on cross-platform impl issues :)

Has anyone spelunked libuv's awesomeness for inspiration on a nice cross-platform implementation style?

general header niceness

https://github.com/joyent/libuv/blob/master/include/uv.h#L62-L67
https://github.com/joyent/libuv/blob/master/include/uv-unix.h
https://github.com/joyent/libuv/blob/master/include/uv-win.h

common timer API

https://github.com/joyent/libuv/blob/master/include/uv.h#L1873-L1881

internal platform specific header timer niceness

https://github.com/joyent/libuv/blob/master/src/unix/internal.h#L180-L181

platform specific impl niceness

https://github.com/joyent/libuv/blob/master/src/win/util.c#L443-L465
https://github.com/joyent/libuv/blob/master/src/unix/core.c#L78-L80
https://github.com/joyent/libuv/blob/master/src/unix/linux-core.c#L245-L249

#9 - 07/24/2013 03:23 AM - kosaki (Motohiro KOSAKI)

(7/20/13 6:39 AM), akr (Akira Tanaka) wrote:

> Issue #8658 has been updated by akr (Akira Tanaka).
>
> File clock_gettime-2.patch added
>
> kosaki (Motohiro KOSAKI) wrote:
>
> > First, Process.times() returns user time and system time and they are process specific. But Process::CLOCK_MONOTONIC is not per-process time.
>
> Yes. Users can choose any clock with Process.clock_gettime unlike other proposals (#8640, #8096).
>
> It seems many people use CLOCK_REALTIME to measure a time interval, though.

So, Why do you choice Process.clock_gettime() instead of Time.clock_gettime()?

> > Second, Linux's CLOCK_MONOTONIC_RAW has the same behavior BSD's CLOCK_MONOTONIC. And, an application which measures a performance need to use CLOCK_MONOTONIC_RAW for avoiding ntp confusing. Then, we should do 1) exporse CLOCK_MONOTONIC_RAW or 2) Process.clock_gettime(Process::CLOCK_MONOTONIC) uses CLOCK_MONOTONIC_RAW internally.
>
> OS specific CLOCK_* constants can be defined.
> Since Process.clock_gettime is a primitive, exchange clk_id is not a good idea.

Hm. OK.

> > Third, using float is a good ruby convention. If we need to use inter (for precision and performance?), the method should have a precision explanation, likes get_time_nanosecond. I mean, ruby interpreter can't warn nor detect following mistake.
> >
> > a = foo # this is usec
> > b = bar # this is nsec
> > c = a + b
> >
> > then, we should warn by method name verbosely. IMHO.
>
> Hm. It is acceptable as far as the exact result (number of nanoseconds) can be obtained.
>
> After thinking while, I find Process.clock_gettime(clk_id, unit).
> unit is an optional argument and :nanoseconds specifies the nanoseconds.
> This can help performance on ILP33 because :microseconds with CLOCK_PROCESS_CPUTIME_ID
> will not use Bignum until 1073 seconds after process start up.
>
> I updated the patch.

An optional argument sound good idea.

thanks.

#10 - 07/24/2013 10:53 AM - akr (Akira Tanaka)

2013/7/24 KOSAKI Motohiro kosaki.motohiro@gmail.com:

> So, Why do you choice Process.clock_gettime() instead of
> Time.clock_gettime()?

I don't like the result value of clock_gettime(CLOCK_REALTIME) because
the value is interpreted differently between systems which use
leapseconds and not.
Time.now should be used instead.

The patch defines Process::CLOCK_REALTIME but it is just for consistency.
I felt definining CLOCK_* constants except CLOCK_REALTIME is too inconsistent.

The expected my main usecase would be CLOCK_PROCESS_CPUTIME_ID (or CLOCK_THREAD_CPUTIME_ID).
For example, I use Process.times for measure Bignum speed but
Process.times cannot measure under 10ms on my environment.
Repeating target operation (as I do) improves precision but
high resolution clocks can be used to obtain similar precision
with less repeatation.

Tanaka Akira

#11 - 07/26/2013 02:53 AM - kosaki (Motohiro KOSAKI)

(7/23/13 9:50 PM), Tanaka Akira wrote:

> 2013/7/24 KOSAKI Motohiro kosaki.motohiro@gmail.com:
>
>> So, Why do you choice Process.clock_gettime() instead of
>> Time.clock_gettime()?
>
> I don't like the result value of clock_gettime(CLOCK_REALTIME) because
> the value is interpreted differently between systems which use
> leapseconds and not.
> Time.now should be used instead.

Hmm. OK.

> The patch defines Process::CLOCK_REALTIME but it is just for consistency.
> I felt definining CLOCK_* constants except CLOCK_REALTIME is too inconsistent.
>
> The expected my main usecase would be CLOCK_PROCESS_CPUTIME_ID (or
> CLOCK_THREAD_CPUTIME_ID).
> For example, I use Process.times for measure Bignum speed but
> Process.times cannot measure under 10ms on my environment.
> Repeating target operation (as I do) improves precision but
> high resolution clocks can be used to obtain similar precision
> with less repeatation.

Really? I don't think so because CLOCK_*_CPUTIME_ID have less precious than
CLOCK_REALTIME. following "t" often show 0 on several OSs.

t0 = clock_gettime(CLOCK_THREAD_CPUTIME_ID)

blah blah blah

t1 = clock_gettime(CLOCK_THREAD_CPUTIME_ID)
t = t1 - t0

#12 - 07/26/2013 12:53 PM - akr (Akira Tanaka)

2013/7/24 Eregon (Benoit Daloze) redmine@ruby-lang.org:

> Issue #8658 has been updated by Eregon (Benoit Daloze).
>
> A very poor one as mapping to Linux/UNIX constants would just confuse people.
> I do not think the UNIX API clock_gettime() for this is the most suitable,
> it does not abstract the functionality and the name/usage is not very ruby-like.

If constants defined by Unix is not suitable, original constants can be defined.

Ruby uses POSIX functions in general.
So I think clock_gettime is very Ruby-ish (and I guess it is easier
than original design
to persuade matz).

> I think FFI would be a good way if someone need direct access to that low-level C function (except for accessing the constants, that would not be handy).

How FFI can be used to call clock_gettime?
I don't have experience with FFI.

> I believe providing a method which is only available in a quite restricted set of platforms is to be avoided.
> In Python it is simply not defined on non-supporting platforms.

At least, CLOCK_REALTIME can be emulated on all platforms.
Other clocks may be too, on some platforms.

Also, Ruby provides many platform dependent methods in Process.

>> Higer level methods may be useful but what I intend in this issue is a
>> low level primitive.

> To which use-cases other than benchmarking do you think?

I expect that I use CLOCK_PROCESS_CPUTIME_ID.

> I want Ruby to propose a nice and precise way to benchmark code *not* requiring the user to know about every detail of available clocks/timers under every platform.

It is good to have such high level methods but
it doesn't conflict with low level methods.

Tanaka Akira

#13 - 07/26/2013 12:59 PM - akr (Akira Tanaka)

2013/7/26 KOSAKI Motohiro kosaki.motohiro@gmail.com:

> Really? I don't think so because CLOCK_*_CPUTIME_ID have less precious than
> CLOCK_REALTIME. following "t" often show 0 on several OSs.
>
> t0 = clock_gettime(CLOCK_THREAD_CPUTIME_ID)
>
> # blah blah blah
>
> t1 = clock_gettime(CLOCK_THREAD_CPUTIME_ID)
> t = t1 - t0

At least, my system can return 1ns unit.
(It doesn't mean 1ns accuracy, of course.)

In following result, 4666 and 4667 are returned.

% ./ruby -ve '
t1 = Process.clock_gettime(Process::CLOCK_PROCESS_CPUTIME_ID, :nanoseconds)
while true
t2 = Process.clock_gettime(Process::CLOCK_PROCESS_CPUTIME_ID, :nanoseconds)
p t2-t1
t1 = t2
end
'|head -20|sort -n
ruby 2.1.0dev (2013-07-19 trunk 42049) [x86_64-linux]
2861
3984
4154
4189
4222
4267
4452
4567
4620
4666

```
4667
4771
4870
4920
5296
6155
6428
12690
20290
-e:5:in p': Broken pipe (Errno::EPIPE) from -e:5:in '
% uname -mrsv
Linux 3.2.0-4-amd64 #1 SMP Debian 3.2.46-1 x86_64
```

Other platforms can behave differently, though.

Tanaka Akira

#14 - 07/27/2013 08:53 AM - akr (Akira Tanaka)

2013/7/26 Tanaka Akira akr@fsij.org:

> 2013/7/26 KOSAKI Motohiro kosaki.motohiro@gmail.com:
>
> > Really? I don't think so because CLOCK_*_CPUTIME_ID have less precious than
> > CLOCK_REALTIME. following "t" often show 0 on several OSs.
> >
> > t0 = clock_gettime(CLOCK_THREAD_CPUTIME_ID)
> >
> > # blah blah blah
> >
> > t1 = clock_gettime(CLOCK_THREAD_CPUTIME_ID)
> > t = t1 - t0
>
> At least, my system can return 1ns unit.
> (It doesn't mean 1ns accuracy, of course.)
>
> In following result, 4666 and 4667 are returned.
>
> ```
> % ./ruby -ve '
> t1 = Process.clock_gettime(Process::CLOCK_PROCESS_CPUTIME_ID, :nanoseconds)
> while true
> t2 = Process.clock_gettime(Process::CLOCK_PROCESS_CPUTIME_ID, :nanoseconds)
> p t2-t1
> t1 = t2
> end
> '|head -20|sort -n
> ```

CLOCK_THREAD_CPUTIME_ID can return 1ns unit too.

```
% ./ruby -ve '
t1 = Process.clock_gettime(Process::CLOCK_THREAD_CPUTIME_ID, :nanoseconds)
while true
t2 = Process.clock_gettime(Process::CLOCK_THREAD_CPUTIME_ID, :nanoseconds)
p t2-t1
t1 = t2
end
'|head -20|sort -n
ruby 2.1.0dev (2013-07-19 trunk 42049) [x86_64-linux]
2091
3913
3936
3939
3975
3976
3978
4281
4466
4473
4516
4563
4666
4706
```

-e:5:in p'5604 6196 6342 13047 19388 : Broken pipe (Errno::EPIPE) from -e:5:in '

(3975 and 3976 is returned.)

CLOCK_THREAD_CPUTIME_ID may be preferable than CLOCK_PROCESS_CPUTIME_ID for measuring Bignum speed because it is purely single thread.

Tanaka Akira

#15 - 08/01/2013 09:10 PM - akr (Akira Tanaka)

- File clock_gettime-3.patch added

I updated the patch to emulate CLOCK_REALTIME using gettimeofday.

#16 - 08/05/2013 09:50 PM - akr (Akira Tanaka)

- File clock_gettime-4.patch added

I updated the patch for Process.clock_gettime.
The patch, clock_gettime-4.patch, supports gettimeofday() and time(),
even when clock_gettime() is available.

#17 - 08/11/2013 12:13 PM - akr (Akira Tanaka)

- Status changed from Open to Closed

I committed r42504 to implement Process.clock_gettime method.
This is result of the meeting:
https://bugs.ruby-lang.org/projects/ruby/wiki/DevelopersMeeting20130809

This doesn't mean that high level API is rejected.
Feel free to discuss it.

#18 - 10/02/2013 03:48 AM - headius (Charles Nutter)

I missed the discussion on this, but here's the summary of JRuby/JVM case:

If monotonic clock is available at OS level, System.nanoTime is equivalent to clock_gettime(CLOCK_MONOTONIC). I suppose there may be some embedded systems or obscure platforms that don't have a monotonic clock, but otherwise I'm guessing this is going to be pretty universal across *nixes. The code in JDK is in http://hg.openjdk.java.net/jdk7/jdk7/hotspot/file/9b0ca45cd756/src/os/linux/vm/os_linux.cpp at line 1453.

If monotonic clock is not available on *nix, JVM will fall back on gettimeofday transparently. I'm investigating whether it is possible for us to query this behavior.

System.currentTimeMillis is just a plain gettimeofday call, which in the Process::clock_gettime API is called GETTIMEOFDAY_BASED_CLOCK_REALTIME.

So JRuby will always be able to support GETTIMEOFDAY_BASED_CLOCK_REALTIME via System.currentTimeMillis, CLOCK_MONOTONIC when nanoTime is monotonic, and the other forms when we're able to make a native downcall. Initially, we will probably just support these two.

I HAVE A QUESTION, however... what about Windows? There's no mention at all in the rdoc about Windows support. I need to investigate what currentTimeMillis and nanoTime do on JVM on Windows.

#19 - 10/02/2013 04:02 AM - headius (Charles Nutter)

JRuby issue for this feature: https://github.com/jruby/jruby/issues/1056

Windows information for JVM:

System.currentTimeMillis is always implemented using win32 GetSystemTimeAsFileTime function.

http://msdn.microsoft.com/en-us/library/windows/desktop/ms724397(v=vs.85).aspx

System.nanoTime is implemented using QueryPerformanceCounter if available (Windows 2000 or higher), falling back on currentTimeMillis if it is not. This appears (by looking around in other articles) to be "rawtime", so perhaps equivalent to CLOCK_MONOTONIC_RAW? I think that would suffice for us to use it for CLOCK_MONOTONIC, but it's not very clear from available info.

http://msdn.microsoft.com/en-us/library/windows/desktop/ms644904(v=vs.85).aspx

The latter article has a link in comments to a HOWTO on implementing a high-resolution timer on Windows. Bottom line is that some synchronization between the two functions is necessary, and it's pretty ugly.

#20 - 10/02/2013 05:13 AM - headius (Charles Nutter)

Implementation in JRuby: https://github.com/jruby/jruby/commit/8c066241bd847b68d8d7255893edbad2d6c311d2

#21 - 10/02/2013 07:29 AM - akr (Akira Tanaka)

2013/10/2 headius (Charles Nutter) headius@headius.com:

> I HAVE A QUESTION, however... what about Windows? There's no mention at all in the rdoc about Windows support. I need to investigate what currentTimeMillis and nanoTime do on JVM on Windows.

usa-san implemented clock_gettime() function in win32/win32.c.

Tanaka Akira

#22 - 10/02/2013 01:29 PM - naruse (Yui NARUSE)

akr (Akira Tanaka) wrote:

> 2013/10/2 headius (Charles Nutter) headius@headius.com:
>
> > I HAVE A QUESTION, however... what about Windows? There's no mention at all in the rdoc about Windows support. I need to investigate what currentTimeMillis and nanoTime do on JVM on Windows.
>
> usa-san implemented clock_gettime() function in win32/win32.c.

It is r42557 and other commits

Files

clock_gettime.patch	4.11 KB	07/19/2013	akr (Akira Tanaka)
clock_gettime-2.patch	7.58 KB	07/20/2013	akr (Akira Tanaka)
clock_gettime-3.patch	8.3 KB	08/01/2013	akr (Akira Tanaka)
clock_gettime-4.patch	9.2 KB	08/05/2013	akr (Akira Tanaka)