Ruby - Feature #3436

Spawn the timer thread lazily

06/13/2010 08:31 PM - mxey (Maximilian Gass)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

=begin
As discussed in http://redmine.ruby-lang.org/issues/show/1820, Ruby 1.9.1
always spawns a timer thread which is required to handle scheduling of Ruby
threads. Rubinius did the same and I suggested to only spawn the thread if it
is required, only when Ruby threading is used. I don't know anything about the
internals of Ruby, but could this method be used in MRI as well?

The timer thread prevents the CPU from going idle and saving power/battery, so it would be nice to avoid it.
=end

Related issues:

Has duplicate Ruby - Bug #3919: Ruby in PowerTOP - too many CPU wakeups	**Closed**	**10/08/2010**

History

#1 - 06/15/2010 06:05 AM - coatl (caleb clausen)

=begin
Unfortunately, that timer thread is also somehow involved with the select statement that lurks at the bottom of ruby's io system like tiamat in the deepest pit of hell. If you get rid of the timer thread, you break io. I don't exactly understand what that code is doing, myself. If I did, I might be able to create a patch that obviates the need for the timer thread at all. Does anyone understand thread_timer sufficiently well to explain it? Why does it call ubf_select_each?
=end

#2 - 10/05/2010 03:09 AM - Spakman (Mark Somerville)

- File bug_3436-spawn_the_timer_thread_lazily.patch added

=begin
I've attached a patch to fix this.

The thread is now only used when it is required to schedule Ruby threads. When there is only the main thread, signals are handled immediately in sighandler().

I only have access to Linux boxes. One of the added tests isn't used on other platforms.

=end

#3 - 10/08/2010 11:13 PM - nobu (Nobuyoshi Nakada)

=begin
Hi,

At Tue, 5 Oct 2010 03:09:42 +0900,
Mark Somerville wrote in [ruby-core:32686]:

> The thread is now only used when it is required to schedule
> Ruby threads. When there is only the main thread, signals are
> handled immediately in sighandler().

Seems almost fine. Since aded missing prototypes, no needs to move some functions now.

> I only have access to Linux boxes. One of the added tests
> isn't used on other platforms.

Updated patch with new test.

```
diff --git c/ext/-test-/thread/timer/extconf.rb w/ext/-test-/thread/timer/extconf.rb
new file mode 100644
index 0000000..a28904b
--- /dev/null
+++ w/ext/-test-/thread/timer/extconf.rb
@@ -0,0 +1,3 @@
+require 'mkmf'
+$INCFLAGS << " -I$(top_srcdir)"
+create_makefile('-test-/thread/timer')
diff --git c/ext/-test-/thread/timer/timer.c w/ext/-test-/thread/timer/timer.c
new file mode 100644
index 0000000..6d77b1c
--- /dev/null
+++ w/ext/-test-/thread/timer/timer.c
@@ -0,0 +1,15 @@
+#include <ruby.h>
+#include "vm_core.h"
+
+VALUE
+thread_timer_thread_running_p(VALUE klass)
+{
```

- return rb_thread_timer_thread_is_running() ? Qtrue : Qfalse;
  +}

```
+void
+Init_timer(void)
+{
```

- VALUE timer = rb_define_module_under(rb_cThread, "Timer");
- rb_define_module_function(timer, "running?", thread_timer_thread_running_p, 0);
  +}
  diff --git c/process.c w/process.c
  index 55b83b1..3a35725 100644
  --- c/process.c
  +++ w/process.c
  @@ -998,7 +998,7 @@ static int forked_child = 0;
  #define before_exec()
  (rb_enable_interrupt(), (void)(forked_child ? 0 : (rb_thread_stop_timer_thread(), 1)))
  #define after_exec() \

- (rb_thread_reset_timer_thread(), rb_thread_start_timer_thread(), forked_child = 0, rb_disable_interrupt())

- (rb_thread_reset_timer_thread(), rb_thread_start_timer_thread(), forked_child = 0)
  #define before_fork() before_exec()
  #define after_fork() (GET_THREAD()->thrown_errinfo = 0, after_exec())

```
diff --git c/signal.c w/signal.c
index ba35954..ce44d81 100644
--- c/signal.c
+++ w/signal.c
@@ -519,6 +519,10 @@ sighandler(int sig)
#if !defined(BSD_SIGNAL) && !defined(POSIX_SIGNAL)
ruby_signal(sig, sighandler);
#endif
+
```

- if (rb_thread_alone()) {
- rb_threadptr_check_signal(GET_THREAD());
- }
  }

```
int
diff --git c/test/ruby/test_signal.rb w/test/ruby/test_signal.rb
index 0098ccc..6371d8b 100644
--- c/test/ruby/test_signal.rb
+++ w/test/ruby/test_signal.rb
@@ -181,4 +181,20 @@ class TestSignal < Test::Unit::TestCase
w.close
assert_equal(r.read, "foo")
end
+
```

- def test_signals_before_and_after_timer_thread

- count = 0
- Signal.trap(:INT) { count += 1 }

- Process.kill :INT, Process.pid
- assert_equal 1, count

- th = Thread.new { sleep 0.5 }
- Process.kill :INT, Process.pid
- assert_equal 2, count

- th.join
- Process.kill :INT, Process.pid
- assert_equal 3, count
- end
  end
  diff --git c/test/thread/test_timer_thread.rb w/test/thread/test_timer_thread.rb
  new file mode 100644
  index 0000000..b732aa0
  --- /dev/null
  +++ w/test/thread/test_timer_thread.rb
  @@ -0,0 +1,10 @@
  +require 'test/unit'
  +require '-test-/thread/timer'

+class TestTimerThread < Test::Unit::TestCase

- def test_timer_is_created_and_destroyed
- assert !Thread::Timer.running?
- assert Thread.new {Thread::Timer.running?}.value
- assert !Thread::Timer.running?
- end
  +end
  diff --git c/thread.c w/thread.c
  index 11c87be..471140e 100644
  --- c/thread.c
  +++ w/thread.c
  @@ -562,6 +562,7 @@ thread_create_core(VALUE thval, VALUE args, VALUE (*fn)(ANYARGS))
  th->status = THREAD_KILLED;
  rb_raise(rb_eThreadError, "can't create Thread (%d)", err);
  }
- rb_thread_start_timer_thread();
  return thval;
  }

@@ -577,6 +578,7 @@ thread_s_new(int argc, VALUE *argv, VALUE klass)
rb_raise(rb_eThreadError, "uninitialized thread - check `%s#initialize'",
rb_class2name(klass));
}

- rb_thread_start_timer_thread();
  return thread;
  }

@@ -711,6 +713,12 @@ thread_join(rb_thread_t *target_th, double delay)
thread_debug("thread_join: success (thid: %p)\n",
(void *)target_th->thread_id);

- rb_disable_interrupt();
- if (rb_thread_alone() && rb_signal_buff_size() == 0) {
- rb_thread_stop_timer_thread();
- }
- rb_enable_interrupt();
- if (target_th->errinfo != Qnil) {
  VALUE err = target_th->errinfo;

@@ -2723,7 +2731,15 @@ void
rb_thread_start_timer_thread(void)
{
system_working = 1;

- rb_thread_create_timer_thread();

- if (!rb_thread_alone()) {
- rb_thread_create_timer_thread();

- }
  +}

+int
+rb_thread_timer_thread_is_running(void)
+{

- return native_timer_thread_is_running();
  }

static int
@@ -4262,7 +4278,7 @@ Init_Thread(void)
}
}

- rb_thread_create_timer_thread();

- rb_thread_start_timer_thread();

  (void)native_mutex_trylock;
  }
  diff --git c/thread_pthread.c w/thread_pthread.c
  index e835bf8..b229b32 100644
  --- c/thread_pthread.c
  +++ w/thread_pthread.c
  @@ -840,6 +840,12 @@ native_reset_timer_thread(void)
  timer_thread_id = 0;
  }

+static int
+native_timer_thread_is_running(void)
+{

- return timer_thread_id != 0;
  +}

#ifdef HAVE_SIGALTSTACK
int
ruby_stack_overflowed_p(const rb_thread_t *th, const void *addr)
diff --git c/thread_win32.c w/thread_win32.c
index 9e64ea4..e52f8de 100644
--- c/thread_win32.c
+++ w/thread_win32.c
@@ -590,4 +590,10 @@ native_reset_timer_thread(void)
}
}

+static int
+native_timer_thread_is_running(void)
+{

- return timer_thread_id != 0;
  +}

#endif /* THREAD_SYSTEM_DEPENDENT_IMPLEMENTATION */
diff --git c/vm_core.h w/vm_core.h
index e43f539..7dc4b8d 100644
--- c/vm_core.h
+++ w/vm_core.h
@@ -635,6 +635,7 @@ VALUE rb_iseq_eval(VALUE iseqval);
VALUE rb_iseq_eval_main(VALUE iseqval);
void rb_enable_interrupt(void);
void rb_disable_interrupt(void);
+int rb_thread_timer_thread_is_running(void);
#if defined **GNUC** && **GNUC** >= 4
#pragma GCC visibility pop
#endif

--
Nobu Nakada

=end

#4 - 10/08/2010 11:36 PM - ko1 (Koichi Sasada)

=begin
(2010/10/08 15:12), Nobuyoshi Nakada wrote:

> Hi,
>
> At Tue, 5 Oct 2010 03:09:42 +0900,
> Mark Somerville wrote in [ruby-core:32686]:
>
> > The thread is now only used when it is required to schedule
> > Ruby threads. When there is only the main thread, signals are
> > handled immediately in sighandler().
>
> Seems almost fine. Since aded missing prototypes, no needs to
> move some functions now.

Timer thread also handles signals.
I'm not sure that your patch work fine.

--
// SASADA Koichi at atdot dot net

=end

#5 - 10/10/2010 01:28 AM - Spakman (Mark Somerville)

=begin
On Fri, Oct 08, 2010 at 11:12:47PM +0900, Nobuyoshi Nakada wrote:

> Updated patch with new test.

The new test is much better, thanks a lot.

On Fri, Oct 08, 2010 at 11:36:13PM +0900, SASADA Koichi wrote:

> (2010/10/08 15:12), Nobuyoshi Nakada wrote:
>
> > At Tue, 5 Oct 2010 03:09:42 +0900,
> > Mark Somerville wrote in [ruby-core:32686]:
> >
> > > The thread is now only used when it is required to schedule
> > > Ruby threads. When there is only the main thread, signals are
> > > handled immediately in sighandler().
> >
> > Seems almost fine. Since aded missing prototypes, no needs to
> > move some functions now.
>
> Timer thread also handles signals.
> I'm not sure that your patch work fine.

The signal handling has not required too many changes - signals are
still added to signal_buff when they are received and in multi-threaded
programs the timer thread is still used to tell the main thread to
process them. In single-threaded programs however, the signals in
signal_buff are processed within sighandler() immediately, since the
timer thread doesn't exist.

I've attached a slightly improved patch which fixes a rare case when the
timer thread may not be stopped correctly after the last thread has
exited.

Attachment: (unnamed)
Attachment: (unnamed)
=end

#6 - 10/10/2010 02:21 AM - Spakman (Mark Somerville)

=begin
On Sun, Oct 10, 2010 at 01:27:53AM +0900, Mark Somerville wrote:

> I've attached a slightly improved patch which fixes a rare case when the timer thread may not be stopped correctly after the last thread has exited.

Oops! Please ignore the previous patch and consider this one instead - in an attempt to trigger the rare bug that I mentioned above, I had altered one of the conditionals, but forgot to change it back. Sorry!

Attachment: (unnamed)
Attachment: (unnamed)
=end

#7 - 10/13/2010 11:18 PM - Spakman (Mark Somerville)

=begin
On Sun, Oct 10, 2010 at 02:21:41AM +0900, Mark Somerville wrote:

> On Sun, Oct 10, 2010 at 01:27:53AM +0900, Mark Somerville wrote:
>
> > I've attached a slightly improved patch which fixes a rare case when the timer thread may not be stopped correctly after the last thread has exited.

How can I move this patch forward? Are there any objections to it being merged? If so, I'd love to try to fix them.

Without the patch, PowerTOP shows that Ruby trunk (-e "sleep 60") is a bad offender when it comes to CPU wakeups:

```
     Wakeups-from-idle per second : 127.2    interval: 15.0s
     no ACPI power usage estimate available
     Top causes for wakeups:
       43.1% ( 99.5)              ruby : hrtimer_start_range_ns
     (hrtimer_wakeup)
       22.6% ( 52.1)     <kernel core> : hrtimer_start_range_ns
     (tick_sched_timer)
        5.5% ( 12.8)       <interrupt> : ahci
        5.2% ( 12.0)     <kernel core> : __mod_timer (rh_timer_func)
        4.7% ( 10.7)     <kernel core> : hrtimer_start
(tick_sched_timer)
        3.8% (  8.8)   chromium-browse : hrtimer_start_range_ns
     (hrtimer_wakeup)
     ...
```

With the patch applied, Ruby doesn't make it onto the (untruncated) list at all:

```
     Wakeups-from-idle per second : 68.9     interval: 15.0s
     no ACPI power usage estimate available
     Top causes for wakeups:
       45.2% ( 50.9)     <kernel core> : hrtimer_start_range_ns
     (tick_sched_timer)
       10.8% ( 12.2)     <kernel core> : __mod_timer (rh_timer_func)
       10.7% ( 12.0)       <interrupt> : ahci
       10.1% ( 11.3)     <kernel core> : hrtimer_start
(tick_sched_timer)
        3.4% (  3.8)   chromium-browse : __mod_timer
(process_timeout)
        2.5% (  2.9)   chromium-browse : hrtimer_start_range_ns
     (hrtimer_wakeup)
     ...
```

I've attached an update that makes the patch apply cleanly again after some changes were made to trunk.

Attachment: (unnamed)
Attachment: (unnamed)
=end

#8 - 10/13/2010 11:41 PM - ko1 (Koichi Sasada)

=begin
(2010/10/13 15:18), Mark Somerville wrote:

> How can I move this patch forward? Are there any objections to it being merged? If so, I'd love to try to fix them.

Yes, I have. Two reasons.

(1) You missed signal problem which cause critical timing bug. Please read thread_timer() on the thread_pthread.c. It is tough for me to describe the behavior in English.

In short, we need to repeat sending a signal to wake-up the target ruby thread completely.

ex) How to wake up the thread?

```
check_signal();
  <- receive signal at this timing
select(..., infinitely);
```

The thread never wake-up. Generally, to avoid this timing issue, pselect(2) is provided. However, select(2) is not only blocking system calls on the Ruby's case.

(2) Performance issue. Your patch start/kill the timer thread. In general, the native thread creation/deletion cause some performance issue.

I understand your issue. In fact, I'm considering this problem and seeking the solution in recent months.

Could you wait for some days? I may show the another idea.

BTW, recent python solve with some clever (complex, for me) method.
http://www.dabeaz.com/python/UnderstandingGIL.pdf

--
// SASADA Koichi at atdot dot net

=end

#9 - 10/15/2010 12:46 AM - Spakman (Mark Somerville)

=begin
Hi Koichi,

On Wed, Oct 13, 2010 at 11:40:52PM +0900, SASADA Koichi wrote:

> It is tough for me to describe the behavior in English.

OK.

> ex) How to wake up the thread?
>
> ```
> check_signal();
>   <- receive signal at this timing
> select(..., infinitely);
> ```

Would this occur if rb_signal_buff_size() > 1 and we process a signal with a handler that calls a blocking function?

> (2) Performance issue. Your patch start/kill the timer thread. In general, the native thread creation/deletion cause some performance issue.

I assumed (perhaps incorrectly) that the overhead was small enough and the creation and deletion uncommon enough that it wouldn't be a problem.

Perhaps there could be a method of increasing the time the timer thread

waits if rb_thread_alone() and creation of a new thread could reduce this again.

> I understand your issue. In fact, I'm considering this problem and seeking the solution in recent months.

It seems to me that we are attempting different things. I have been primarily concerned with reducing/removing CPU wakeups in the single-threaded case and falling back to the timer thread when multi-threaded. Assuming, I understand you correcly, you appear to be trying to remove the timer thread completely.

If you are trying to remove the timer thread altogether, that's *great* news and I won't bother carrying this patch on!

> Could you wait for some days? I may show the another idea.

Sure. I'll be interested to see it!

> BTW, recent python solve with some clever (complex, for me) method.
> http://www.dabeaz.com/python/UnderstandingGIL.pdf

Really interesting reading.

Thanks a lot, very useful!

Attachment: (unnamed)
=end

#10 - 10/15/2010 01:10 AM - ko1 (Koichi Sasada)

=begin
Hi,

(2010/10/14 16:45), Mark Somerville wrote:

> ex) How to wake up the thread?
>
> ```
> check_signal();
>   <- receive signal at this timing
> select(..., infinitely);
> ```

> Would this occur if rb_signal_buff_size() > 1 and we process a signal with a handler that calls a blocking function?

I can't understand your situation.

My situation is occur if signal received at pointed by :

thread.c:2504
BLOCKING_REGION({
/* !!! !!! */
result = select(n, read, write, except, timeout);
if (result < 0) lerrno = errno;
}, ubf_select, GET_THREAD());

is
"after signal checking" and
"before calling system call".

> (2) Performance issue. Your patch start/kill the timer thread. In general, the native thread creation/deletion cause some performance issue.

> I assumed (perhaps incorrectly) that the overhead was small enough and the creation and deletion uncommon enough that it wouldn't be a problem.

I agree that is it not huge performance issue.

> Perhaps there could be a method of increasing the time the timer thread waits if rb_thread_alone() and creation of a new thread could reduce this again.

My idea is keep the timer thread sleep infinity while only one thread is running.

> I understand your issue. In fact, I'm considering this problem and seeking the solution in recent months.

> It seems to me that we are attempting different things. I have been primarily concerned with reducing/removing CPU wakeups in the single-threaded case and falling back to the timer thread when multi-threaded. Assuming, I understand you correcly, you appear to be trying to remove the timer thread completely.

I think we are aimed to solve same problem. I also want to reduce CPU wake up counts. And currently I think we can't remove timer thread.

--
// SASADA Koichi at atdot dot net

=end

#11 - 11/11/2010 02:24 AM - alvherre (Alvaro Herrera)

=begin
May I point you to the solution that PostgreSQL implemented to solve this exact problem? It is here:

http://git.postgresql.org/gitweb?p=postgresql.git;a=tree;f=src/backend/port;hb=HEAD

See the files unix_latch.c and win32_latch.c. They both implement the interface here:

http://git.postgresql.org/gitweb?p=postgresql.git;a=blob;f=src/include/storage/latch.h;hb=HEAD

This implementation is very efficient and very portable.
=end

#12 - 11/11/2010 02:25 AM - alvherre (Alvaro Herrera)

=begin
Forgoot to mention: since this is under a MIT-like license, you could simply lift this code and integrate it into Ruby.
=end

#13 - 11/11/2010 02:55 AM - ko1 (Koichi Sasada)

=begin
(2010/11/11 2:24), Alvaro Herrera wrote:

> May I point you to the solution that PostgreSQL implemented to solve this exact problem? It is here:

What do you mean "this exact problem"? Safety signal treatment?

I see the source code briefly, I can't understand which problem.
(WaitLatchOrSocket() seems to solve only a problem around "select()")

--
// SASADA Koichi at atdot dot net

=end

#14 - 11/11/2010 04:51 AM - alvherre (Alvaro Herrera)

=begin
Oh, now I see that I misread what you were saying. When you mentioned pselect() I thought that you meant that would have solved your problem but you couldn't use it for some reason. Now I realize that you meant that you need to handle system calls other than select(), and so pselect() alone is not enough.

It also doesn't help that this code is written to work on a multi-process environment where threads are forbidden. Since your code is thread-centric, it would take some work to adapt, or perhaps it's downright impossible.

So please ignore me :-)
=end

#15 - 03/07/2011 07:26 PM - Spakman (Mark Somerville)

=begin
Koichi has proposed a potential solution to this problem in a ruby-core thread[1].

It would be *great* to get some more testing and feedback on this, since it seems to be a problem for a number of users.

[1] - http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/33456
=end

#16 - 03/07/2011 08:23 PM - Spakman (Mark Somerville)

=begin
I apologise in advance if this is a duplicate message. I previously
posted this on redmine.ruby-lang.org, but immediately received mail
delivery errors regarding it. I simply don't have time to investigate
that.

It was meant to be a reply to the first post in this thread, but I can't
find that so here goes again:

Koichi has proposed a potential solution to this problem in a ruby-core
thread[1].

[1] - http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/33456
=end

#17 - 04/20/2011 10:12 PM - sunaku (Suraj Kurapati)

=begin
It seems Mark was the only person to review ((<Ko1's patch|URL:http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/33456>)).

What do the other Ruby developers think about it?

Will it not be committed because it lacks Windows support?

Thanks for your consideration.
=end

#18 - 05/10/2011 08:53 AM - Spakman (Mark Somerville)

I'm repeating what I said in another thread to catch people that have subscribed to this bug.

Suraj Kurapati wrote:

> It seems Mark was the only person to review ((<Ko1's patch|URL:http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/33456>)).
>
> What do the other Ruby developers think about it?
>
> Will it not be committed because it lacks Windows support?
>
> Thanks for your consideration.

With the 1.9.2 release schedule now public and rather imminent, now is the time to get this tested and reviewed.

I really think this regression should be fixed for 1.9.3 - if you do too, please help make it happen!

#19 - 06/29/2011 10:44 PM - kosaki (Motohiro KOSAKI)

- Status changed from Open to Closed

Fixed by r32244.

#20 - 07/01/2011 12:23 AM - Spakman (Mark Somerville)

On Wed, Jun 29, 2011 at 10:44:59PM +0900, Motohiro KOSAKI wrote:

> Issue #3436 has been updated by Motohiro KOSAKI.
>
> Status changed from Open to Closed

Fixed by r32244.

Thank you very much and well done to everyone involved - it's working nicely for me (F14, x86_64).

Files

bug_3436-spawn_the_timer_thread_lazily.patch	4.9 KB	10/05/2010	Spakman (Mark Somerville)
noname	207 Bytes	03/07/2011	Spakman (Mark Somerville)
noname	207 Bytes	04/12/2011	Spakman (Mark Somerville)
noname	207 Bytes	04/12/2011	Spakman (Mark Somerville)
noname	207 Bytes	07/01/2011	Spakman (Mark Somerville)