Ruby master - Feature #7086

ConditionVariable#wait has meaningless return value

09/30/2012 12:14 AM - rklemme (Robert Klemme)

Status:	Assigned	
Priority:	Normal	
Assignee:	kosaki (Motohiro KOSAKI)	
Target version:		

Description

I consider this an API bug: when invoked with a timeout value the caller cannot distinguish from the return value whether the condition has been signaled or the time has run out. Consider how Java does it: http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/Condition.html#await(long,%20java.util.concurrent.TimeUnit) and http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/Condition.html#awaitUntil(java.util.Date)

There's another issue exhibited through the script but I will create a separate ticket for this.

History

#1 - 09/30/2012 12:21 AM - rklemme (Robert Klemme)

rklemme (Robert Klemme) wrote:

> There's another issue exhibited through the script but I will create a separate ticket for this.

That is bug #7087 now.

#2 - 09/30/2012 12:47 AM - kosaki (Motohiro KOSAKI)

I think Java API spec is crazy and buggy. It explicitly says:

http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/Condition.html#await(long,%20java.util.concurrent.TimeUnit)

> When waiting upon a Condition, a "spurious wakeup" is permitted to occur, in general, as a concession to the underlying platform semantics.

And then, no timeout doesn't mean the condition become true and an API user always have to ignore the return value and check his own condition again.

Moreover, we have one another mess. When cv#signal was not only fired but also timeout occur, some OS return timeout occur and other return cv#signal received. Both POSIX and Java permit such platform dependency. then, you must not trust the return value.

I'm hesitate to implement it because it seems bring a lot of trouble than worth.

#3 - 09/30/2012 03:44 AM - rklemme (Robert Klemme)

kosaki (Motohiro KOSAKI) wrote:

> I think Java API spec is crazy and buggy.

:-)

> And then, no timeout doesn't mean the condition become true and an API user always have to ignore the return value and check his own condition again.

There is some truth in what you write. In the usual case you certainly need to check your condition. But I think nevertheless that the Java way is not crazy.

> Moreover, we have one another mess. When cv#signal was not only fired but also timeout occur, some OS return timeout occur and other return cv#signal received. Both POSIX and Java permit such platform dependency. then, you must not trust the return value.

I disagree that the named facts make the return value useless. See below.

> I'm hesitate to implement it because it seems bring a lot of trouble than worth.

We certainly need to give a bit more thought to this. So, the condition needs to be checked to be sure it is met. But: in case of timeout you can do a quick exit of the loop if #wait has a proper return value. The usage pattern would look like this:

```
def do_whatever(timeout)
target = Time.now + timeout

lock.synchronize do
until condition
cv.wait(target - Time.now) or return :fail
end

whatever

end

:ok
end
```

The race condition between spurious wakeup, timeout and signaling is really irrelevant. The reason is this: if any two of them happen at approximately the same time it doesn't matter whether one of them is a few microseconds earlier or the implementation prefers one of them. The outcome (i.e. return value) is random for the observer anyway. You also cannot create a test which would verify certain behavior reliably because you cannot control execution order down to the nanosecond.

But: if the timeout is detected it is extremely useful to indicate that fact via the return value. Otherwise the idiom shown above would become more complex:

```
def do_whatever(timeout)
target = Time.now + timeout

lock.synchronize do
until condition
sec = target - Time.now
return :fail if sec <= 0
cv.wait sec
end

whatever

end

:ok
end
```

If we allow instances of Time and DateTime as "timeout" argument to #wait things become even simpler:

```
def do_whatever(timeout)
target = Time.now + timeout

lock.synchronize do
until condition
cv.wait target or return :fail
end

whatever

end

:ok
end
```

It would also be nice if MonitorMixin::ConditionVariable allowed for a timeout argument to #wait_until and #wait_while. Then we could remove the loop altogether yet retain the time limitation. :-)

#4 - 11/05/2012 08:44 PM - mame (Yusuke Endoh)

- Tracker changed from Bug to Feature

- Assignee set to kosaki (Motohiro KOSAKI)

This is not a bug. As you said ("Otherwise the idiom shown above would become more complex"), there is a trivial workaround.
I think that Ruby thread APIs are modeled on pthread APIs. pthread_cond_wait does not return such as information.
So, I'm moving this ticket to feature tracker.

Java thread API may be more appropriate for Ruby because Java features have much in common with Ruby, such as OO and exceptions.

Though, Java threads also have a dark chapter in the history:

http://docs.oracle.com/javase/1.5.0/docs/guide/misc/threadPrimitiveDeprecation.html

--
Yusuke Endoh mame@tsg.ne.jp

#5 - 11/05/2012 08:44 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Target version set to 2.6

#6 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)

Files

timeout-rk.rb	2.51 KB	09/30/2012	rklemme (Robert Klemme)