Ruby - Feature #6647

Exceptions raised in threads should be logged

06/26/2012 07:21 AM - headius (Charles Nutter)

Status:	Closed
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	

Description

Many applications and users I have dealt with have run into bugs due to Ruby's behavior of quietly swallowing exceptions raised in threads. I believe this is a bug, and threads should always at least log exceptions that bubble all the way out and terminate them.

The implementation should be simple, but I'm not yet familiar enough with the MRI codebase to provide a patch. The exception logging should be logged in the same way top-level exceptions get logged, but perhaps with information about the thread that was terminated because of the exception.

Here is a monkey patch that simulates what I'm hoping to achieve with this bug:

```
class << Thread
  alias old_new new

  def new(*args, &block)
    old_new(*args) do |*bargs|
      begin
        block.call(*bargs)
      rescue Exception => e
        raise if Thread.abort_on_exception || Thread.current.abort_on_exception
        puts "Thread for block #{block.inspect} terminated with exception: #{e.message}"
        puts e.backtrace.map {|line| "  #{line}"}
      end
    end
  end
end

Thread.new { 1 / 0 }.join
puts "After thread"
```

Output:

```
system ~/projects/jruby $ ruby thread_error.rb
Thread for block #<Proc:0x000000010d008a80@thread_error.rb:17> terminated with exception: divided
by 0
  thread_error.rb:17:in `/'
  thread_error.rb:17
  thread_error.rb:7:in `call'
  thread_error.rb:7:in `new'
  thread_error.rb:5:in `initialize'
  thread_error.rb:5:in `old_new'
  thread_error.rb:5:in `new'
  thread_error.rb:17
After thread
```

Related issues:

Related to Ruby - Feature #13006: backtrace of thread killer	**Open**

Associated revisions

Revision 2e71c752787e0c7659bd5e89b6c5d433eddfe13a - 06/06/2016 12:25 AM - nobu (Nobuyoshi Nakada)

Thread.report_on_exception

- thread.c (thread_start_func_2): report raised exception if report_on_exception flag is set. [Feature #6647]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@55290 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 2e71c752 - 06/06/2016 12:25 AM - nobu (Nobuyoshi Nakada)

Thread.report_on_exception

- thread.c (thread_start_func_2): report raised exception if
  report_on_exception flag is set. [Feature #6647]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@55290 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 06/26/2012 07:24 AM - headius (Charles Nutter)

FWIW, precedent: Java threads log their exceptions by default. I have *never* found the feature to be a bother, and it makes it nearly impossible to ignore fatally-flawed thread logic that spins up and fails lots of threads.

#2 - 06/26/2012 07:44 AM - rue (Eero Saynatkari)

headius (Charles Nutter) wrote:

> Many applications and users I have dealt with have run into bugs due to Ruby's behavior of quietly swallowing exceptions raised in threads. I believe this is a bug, and threads should always at least log exceptions that bubble all the way out and terminate them.

I have had to set .abort_on_exception more times than I care to remember.

```
rescue Exception => e
  raise if Thread.abort_on_exception || Thread.current.abort_on_exception
  puts "Thread for block #{block.inspect} terminated with exception: #{e.message}"
  puts e.backtrace.map {|line| "  #{line}"}
```

$stderr/warn, but this would improve the current situation significantly.

Can significant upgrade problems be expected if .abort_on_exception defaulted to true? This would seem to be the behaviour to suit most users.

#3 - 06/26/2012 06:23 PM - regularfry (Alex Young)

On 25/06/12 23:44, rue (Eero Saynatkari) wrote:

> Issue #6647 has been updated by rue (Eero Saynatkari).
>
> headius (Charles Nutter) wrote:
>
> > Many applications and users I have dealt with have run into bugs due to Ruby's behavior of quietly swallowing exceptions raised in threads. I believe this is a bug, and threads should always at least log exceptions that bubble all the way out and terminate them.
>
> I have had to set .abort_on_exception more times than I care to remember.

Agreed. It's one of the things I check for in code review. Consider
this a +1 from me.

```
rescue Exception =>  e
  raise if Thread.abort_on_exception || Thread.current.abort_on_exception
  puts "Thread for block #{block.inspect} terminated with exception: #{e.message}"
  puts e.backtrace.map {|line| "  #{line}"}
```

> $stderr/warn, but this would improve the current situation significantly.
>
> Can significant upgrade problems be expected if .abort_on_exception defaulted to true? This would seem to be the behaviour to suit most users.

That sounds a little extreme, although I wouldn't object. I'd be happy
with them not being silently swallowed.

--
Alex

> Bug #6647: Exceptions raised in threads should be logged
> https://bugs.ruby-lang.org/issues/6647#change-27456

Author: headius (Charles Nutter)
Status: Open
Priority: Normal
Assignee:
Category:
Target version:
ruby -v: head

Many applications and users I have dealt with have run into bugs due to Ruby's behavior of quietly swallowing exceptions raised in threads. I believe this is a bug, and threads should always at least log exceptions that bubble all the way out and terminate them.

The implementation should be simple, but I'm not yet familiar enough with the MRI codebase to provide a patch. The exception logging should be logged in the same way top-level exceptions get logged, but perhaps with information about the thread that was terminated because of the exception.

Here is a monkey patch that simulates what I'm hoping to achieve with this bug:

```
class<<  Thread
alias old_new new

def new(*args,&block)
old_new(*args) do |*bargs|
begin
block.call(*bargs)
rescue Exception =>  e
raise if Thread.abort_on_exception || Thread.current.abort_on_exception
puts "Thread for block #{block.inspect} terminated with exception: #{e.message}"
puts e.backtrace.map {|line| "  #{line}"}
end
end
end
end

Thread.new { 1 / 0 }.join
puts "After thread"
```

END

Output:

```
system ~/projects/jruby $ ruby thread_error.rb
Thread for block #Proc:0x000000010d008a80@thread_error.rb:17  terminated with exception: divided by 0
thread_error.rb:17:in /' thread_error.rb:17 thread_error.rb:7:in call'
thread_error.rb:7:in new' thread_error.rb:5:in initialize'
thread_error.rb:5:in old_new' thread_error.rb:5:in new'
thread_error.rb:17
After thread
```

#4 - 06/27/2012 11:23 AM - normalperson (Eric Wong)

Alex Young alex@blackkettle.org wrote:

> On 25/06/12 23:44, rue (Eero Saynatkari) wrote:
>
> > Issue #6647 has been updated by rue (Eero Saynatkari).
> >
> > $stderr/warn, but this would improve the current situation significantly.
> >
> > Can significant upgrade problems be expected if .abort_on_exception defaulted to true? This would seem to be the behaviour to suit most users.
>
> That sounds a little extreme, although I wouldn't object. I'd be
> happy with them not being silently swallowed.

I think aborting the whole process is extreme (though, I usually do
it myself).

I would very much like to see this via $stderr/warn, though.

#5 - 06/27/2012 08:24 PM - rue (Eero Saynatkari)

normalperson (Eric Wong) wrote:

> Alex Young alex@blackkettle.org wrote:
>
> > On 25/06/12 23:44, rue (Eero Saynatkari) wrote:
> >
> > > Issue #6647 has been updated by rue (Eero Saynatkari).
> > >
> > > $stderr/warn, but this would improve the current situation significantly.
> > >
> > > Can significant upgrade problems be expected if .abort_on_exception defaulted to true? This would seem to be the behaviour to suit most users.
> >
> > That sounds a little extreme, although I wouldn't object. I'd be happy with them not being silently swallowed.
>
> I think aborting the whole process is extreme (though, I usually do it myself).

You are probably correct. Reconsidering the issue, the benefit of raising is probably not enough to offset that, thus leaving the $stderr/warn as the better choice.

--
Eero

#6 - 07/14/2012 06:11 PM - nahi (Hiroshi Nakamura)

- Category set to core

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

- Target version set to 2.0.0

Discussions ad CRuby dev meeting at 14th July.

- We can understand the requirement. (We understood that the requirement is dumping something without raising when Thread#abort_on_exception = false)
- Writing to STDERR could cause problem with existing applications so we should take care about it.
- rb_warn() instead of puts would be good because we already using rb_warns.

Matz, do you mind if we dump Thread error with rb_warn if Thread#abort_on_exception = false?

#7 - 09/11/2012 02:44 AM - headius (Charles Nutter)

Any update on this?

#8 - 10/15/2012 05:03 AM - headius (Charles Nutter)

Ping! This came up in JEG's talk at Aloha RubyConf as a recommendation (specifically, set abort_on_exception globally to ensure failed threads don't quietly disappear). Ruby should not allow threads to quietly fail.

#9 - 10/15/2012 11:23 AM - kosaki (Motohiro KOSAKI)

I think "exception raised" callback is better way because an ideal output (both format and output device) depend on an application. It should be passed a raised exception.

#10 - 10/15/2012 06:23 PM - regularfry (Alex Young)

On 15/10/12 03:24, kosaki (Motohiro KOSAKI) wrote:

> Issue #6647 has been updated by kosaki (Motohiro KOSAKI).
>
> I think "exception raised" callback is better way because an ideal output (both format and output device) depend on an application. It should be passed a raised exception.

This, along with a sensible default that displays *something* to stderr, would be absolutely ideal from my point of view.

--

Alex

#11 - 10/18/2012 03:00 AM - headius (Charles Nutter)

I started prototyping a callback version and ran into some complexities I could not easily resolve:

- How does abort_on_exception= interact with a callback system?
  ** I tried implementing abort_on_exception=true to use a builtin callback that raises in Thread.main, but should abort_on_exception=true blow away a previously-set callback?
  ** Similarly: should abort_on_exception=false reset to a do-nothing callback?
  ** If neither of these, how do we combine callback and abort_on_exception behavior?
- Seems like there should be a Thread.default_exception_handler you can set once for all future threads.

My concern is that bikeshedding a callback API -- as useful as it might be -- will cause further delays in the more important behavior of having threads report that they terminated due to an exception.

#12 - 11/17/2012 12:40 AM - headius (Charles Nutter)

Ping!

#13 - 11/25/2012 02:18 AM - headius (Charles Nutter)

Checking in on this again. Can we at least agree it should happen for 2.0.0? Perhaps Matz should review this?

#14 - 11/25/2012 09:46 AM - mame (Yusuke Endoh)

- Tracker changed from Bug to Feature

headius (Charles Nutter) wrote:

> Can we at least agree it should happen for 2.0.0?

No, objection. This looks to me nothing except for a feature request.
I cannot estimate the impact how writing to stderr affects existing applications.
There is a workaround.
I don't think that your concern is so significant that we should address by changing the spec from now.

Moving to the feature tracker and setting to next minor.

--
Yusuke Endoh mame@tsg.ne.jp

#15 - 11/25/2012 09:46 AM - mame (Yusuke Endoh)

- Target version changed from 2.0.0 to 2.6

#16 - 09/27/2013 08:18 PM - headius (Charles Nutter)

So, can we do this for 2.1? I have heard from many other users that really would like exceptions bubbling out of threads to be reported in some way. We have had numerous bug reports relating to code where threads disappear without a trace.

#17 - 09/30/2013 07:53 AM - avdi (Avdi Grimm)

This would indeed eliminate a huge amount of confusion for people getting
started with threads. Or for people years of experience with threads, for
that matter...

--
Avdi Grimm
http://avdi.org

I only check email twice a day. to reach me sooner, go to
http://awayfind.com/avdi

#18 - 09/30/2013 12:20 PM - ko1 (Koichi Sasada)

- Target version changed from 2.6 to 2.1.0

#19 - 09/30/2013 12:23 PM - ko1 (Koichi Sasada)

(2013/09/27 20:18), headius (Charles Nutter) wrote:

> So, can we do this for 2.1? I have heard from many other users that really would like exceptions bubbling out of threads to be reported in some way. We have had numerous bug reports relating to code where threads disappear without a trace.

I'll ask matz.

Does JRuby log it already?
Any problem on your experience if you have?

--
// SASADA Koichi at atdot dot net

#20 - 09/30/2013 11:05 PM - headius (Charles Nutter)

We do not currently log it, but the patch to do so is trivial.

https://gist.github.com/6764310

I'm running tests now to confirm it doesn't break anything.

#21 - 09/30/2013 11:37 PM - headius (Charles Nutter)

Testing seems to indicate this is a pretty safe change, and it just makes the debug-logged exception output be logged any time abort_on_exception is not true.

#22 - 10/02/2013 11:05 AM - akr (Akira Tanaka)

In the yesterday's meeting,
https://bugs.ruby-lang.org/projects/ruby/wiki/DevelopersMeeting20131001Japan
we discussed this issue.

We found that message at thread exiting with exception have a problem.
The thread can be joined after exit and the exception may be handled by joined thread.

```
% ruby -e '
t = Thread.new {
  raise "foo"
}
sleep 1 # the thread exits with an exception.
begin
  t.join
rescue
  p $! # something to do with the exception
end
'
#<RuntimeError: foo>
```

If thread exiting with exception outputs a message,
there is no way to disable to it.

So, the message should be delayed until Ruby is certain that
the thread is not joined.
This means the message should be output at the thread is collected by GC.

#23 - 10/02/2013 02:30 PM - headius (Charles Nutter)

akr (Akira Tanaka) wrote:

> In the yesterday's meeting,
> https://bugs.ruby-lang.org/projects/ruby/wiki/DevelopersMeeting20131001Japan
> we discussed this issue.
>
> We found that message at thread exiting with exception have a problem.
> The thread can be joined after exit and the exception may be handled by joined thread.
> ...
> If thread exiting with exception outputs a message,
> there is no way to disable to it.
>
> So, the message should be delayed until Ruby is certain that
> the thread is not joined.
> This means the message should be output at the thread is collected by GC.

GC is a pretty fuzzy time boundary, but it's not terrible. Handling it will mean some finalization requirement for threads to say "hey, I just GCed this thread that died due to an unhandled exception". I feel like something more explicit is needed.

I guess I need to think about this. Some of the cases I want to fix -- where threads are spun up and left to do their own work -- this might be acceptable. But many users will keep references to worker threads they start in order to explicitly stop them on shutdown or other events. In those

cases, the thread will be hard referenced and never GCed...and there will be no indication that the thread has died.

Perhaps this could be an on-by-default flag? It would require very little work to add something like:

```
class Thread
  def report_on_exception=(report) ..
end
```

...where the default is true. Going forward, this would be like having the debug output of a thread-killing exception always happen, but you could turn it off. That would address your concern about not being able to silence it.

The workflow would go like this:

If you are spinning up a thread to do background work and don't plan to check on it...

- Spin up the thread
- Store it in a list if you like
- A message will be reported if the thread dies in an exceptional way

If you are spinning up a thread you plan to join on at some time in the future...

- Spin up the thread
- Set Thread#report_on_exception = false
- Join at your leisure...no message will be reported

This at least allows users to say "I mean to pick up this thread's results later...don't report an error" without having hard-referenced threads die silently.

Is this a reasonable compromise?

#24 - 10/02/2013 05:05 PM - ko1 (Koichi Sasada)

FYT:
On pthread, there is pthread_detach() which declares nobody join on this thread.
In other words, pthread_detach() is same as Thread#report_on_exception=true.

#25 - 10/02/2013 05:13 PM - ko1 (Koichi Sasada)

Sorry, it is not same, but we can consier that.

BTW, I think it true as default is good idea.

IMO, inter-thread communication via exception with Thread#join should be bad idea.

#26 - 10/03/2013 04:38 AM - headius (Charles Nutter)

ko1 (Koichi Sasada) wrote:

> Sorry, it is not same, but we can consier that.
>
> BTW, I think it true as default is good idea.

So to summarize:

- Exceptions will log when they bubble out of a thread, as with -d, unless Thread#report_on_exception == false
- Thread#report_on_exception defaults to true

Can we do this for 2.1?

> IMO, inter-thread communication via exception with Thread#join should be bad idea.

+1

I had originally wanted something similar to Java, where you can set an "unhandled exception handler" for any thread. That would cover all cases, and the default case would be to just report the error. I was unsuccessful in specifying it because I wasn't sure how it should interact with abort_on_exception=.

#27 - 01/30/2014 06:16 AM - hsbt (Hiroshi SHIBATA)

- Target version changed from 2.1.0 to 2.2.0

#28 - 01/31/2014 07:23 AM - ko1 (Koichi Sasada)

Restart for 2.2.
Matz, do you have any idea?

#29 - 05/08/2015 05:26 PM - godfat (Lin Jen-Shin)

Not sure if a +1 would do anything, but I like the idea of
Thread#report_on_exception defaults to true.

For quick and one time scripts, it's tedious to write
Thread.current.abort_on_exception = true all the time,
and it shouldn't be set to true by default, either.
So at least make debugging easier by default is a good idea,
and who doesn't like to see warnings anyway? :P

I was referred from yahns mailing list:
http://yhbt.net/yahns-public/m/20150508170311.GA1260%40dcvr.yhbt.net.html
Which some worker threads were dead silently and it's puzzling
if I don't even know there's an exception was raised.

#30 - 05/08/2015 09:48 PM - normalperson (Eric Wong)

I have an actual patch which is only 2 lines, but there's some test
failures and MANY warnings I don't feel motivated to fix just yet
unless matz approves the feature:

http://80x24.org/spew/m/0a12f5c2abd2dfc2f055922a16d02019ee707397.txt

#31 - 05/09/2015 12:40 AM - headius (Charles Nutter)

Eric Wong wrote:

> I have an actual patch which is only 2 lines, but there's some test
> failures and MANY warnings I don't feel motivated to fix just yet
> unless matz approves the feature:

Hot diggity! I bet there's several of these that indicate bugs to be fixed. At the very least, they indicate exceptions that are being raised and not dealt with.

I think this is great evidence that this IS the right change to make.

#32 - 03/31/2016 03:26 PM - Eregon (Benoit Daloze)

Akira Tanaka wrote:

> In the yesterday's meeting,
> https://bugs.ruby-lang.org/projects/ruby/wiki/DevelopersMeeting20131001Japan
> we discussed this issue.
>
> We found that message at thread exiting with exception have a problem.
> The thread can be joined after exit and the exception may be handled by joined thread.
>
> ```
> % ruby -e '
> t = Thread.new {
>   raise "foo"
> }
> sleep 1 # the thread exits with an exception.
> begin
>   t.join
> rescue
>   p $! # something to do with the exception
> end
> '
> #<RuntimeError: foo>
> ```
>
> If thread exiting with exception outputs a message,
> there is no way to disable to it.
>
> So, the message should be delayed until Ruby is certain that
> the thread is not joined.
> This means the message should be output at the thread is collected by GC.

I am strongly in favor of having something like Thread#report_on_exception, defaulting to true.
If a Thread can support known exceptions, it can rescue them explicitly.

If a Thread is used as some sort of isolation, it can disable #report_on_exception.

Thread#join is not enough, and because of the lack of reporting it's very easy to end in a deadlock with no other way to notice than dumping the thread stacks.
Imagine a simple actor framework where actors are just Threads with a Queue.
Actor1 waits for a message from Actor2, and Actor2 crashes because for instance it calls a method which does not exist.
Actor1 is blocked, and the user has absolutely no knowledge of what's happening, unless Thread.abort_on_exception is set before creating any thread.

So, in summary, Thread.abort_on_exception is not always appropriate,
Thread#join is not enough,
and silently swallowing exceptions can lead to deadlocks that the programmer has a hard time to notice.
Let's give a chance to users to see problems in their code with Thread!

#33 - 04/28/2016 07:02 AM - shyouhei (Shyouhei Urabe)

I remember this topic was looked at in the developer meeting this month. Matz was positive to have Thread#report_on_exception, but not default true. Sorry I don't remember the reason why he was not comfortable with defaulting this.

#34 - 04/29/2016 11:40 AM - Eregon (Benoit Daloze)

Shyouhei Urabe wrote:

> I remember this topic was looked at in the developer meeting this month. Matz was positive to have Thread#report_on_exception, but not default true.

Thanks for the reply.

> Sorry I don't remember the reason why he was not comfortable with defaulting this.

That's unfortunate.
The main reason to have it by default is to give a chance when developing with Threads to notice the error in the sub-thread.
When trying out threads, the program will of course have very little chance to have Thread.abort_on_exception or Thread.report_on_exception in it, particularly if the author is not extremely familiar with current Ruby thread exception pitfalls.
Debug mode (-d) would help but it's also a feature most people ignore or do not think to use (and it outputs much more).

That's why I think report_on_exception by default is the only reasonable choice for people not extremely familiar with Ruby thread and their exception handling.

I would guess the argument is about Thread#join or Thread#value.
But threads in Ruby are OS threads, using them for just one computation (like a future) is inefficient as it incurs spawning a OS thread every time.
For this and many other reasons, joining threads is often done much later than when the exception happens (if ever, for instance with a dead/livelock it would not).
Here is another example: Communication with threads is done most often with Queue, yet if the main Thread pops from the queue to get results from the sub-thread (producer) and the sub-thread throws an exception, the program will deadlock with no clue given to the programmer.

So relying on #join or #value is very brittle and I believe causes much more harm than a few extra exceptions printed on stdout, which can be easily handled with Thread.current.report_on_exception = false.

#35 - 05/02/2016 04:38 PM - headius (Charles Nutter)

> I remember this topic was looked at in the developer meeting this month. Matz was positive to have Thread#report_on_exception, but not default true. Sorry I don't remember the reason why he was not comfortable with defaulting this.

I would guess it's for all the badly-behaved code out there that's just letting threads die silently when an error is raised.

Having it default to off defeats the purpose of this feature request. My request is that threads report when an exception bubbles out before being handled.

I understand the concern about Thread#join. If we report by default then we might have exceptions reported as unhandled when a subsequent Thread#join would handle them. The idea about reporting on GC of the thread is interesting but it might mean we still never get any indication if the thread never GCs. I'd expect this is the typical case, since most people don't fire off threads without having a hard reference to them.

Re: Thread#join

Thread#join always re-raises its exception no matter whether abort_on_exception is set, so I don't see this as an issue. If you expect you'll be handling a Thread's last exception via #join, you would just specify that it should be quiet.

Thread#join works this way right now for abort_on_exception:

```
2.3.0 :001 > Thread.abort_on_exception = true
 => true
2.3.0 :002 > go = false
 => false
2.3.0 :003 > t = Thread.new { Thread.pass until go; raise }
 => #<Thread:0x007fc8920abe98@(irb):3 run>
2.3.0 :004 > begin; go = true; sleep; rescue Exception; p $!; end
RuntimeError
 => RuntimeError
2.3.0 :005 > t.join
RuntimeError:
 from (irb):3:in `block in irb_binding'
```

I also made the larger suggestion of having this all wire in as a per-thread exception handler API.

There would be at least four default handlers. In all cases I think #join and #value should still produce the original exception.

- Silent: Do not propagate the exception and do not report it. This is current behavior.
- Report: Do not propagate the exception but report that it ended a thread. This is my requested behavior.
- Reraise: Propagate the exception to the main thread but do not otherwise report it. This is abort_on_exception = true.
- Custom: Provide your own proc/block to handle any exception raised from the thread.

Note also we could blunt some compatibility concerns by making these settable as Thread.new keyword args:

t = Thread.new(exception: :raise)

#36 - 05/17/2016 05:53 AM - hsbt (Hiroshi SHIBATA)

- Description updated

#37 - 05/17/2016 05:54 AM - nobu (Nobuyoshi Nakada)

- Description updated

#38 - 05/17/2016 05:58 AM - akr (Akira Tanaka)

How about introducing Thread[.#]report_on_exception=(bool) ?

false by default for compatibility.
message should be printed at thread exit (not GC).

#39 - 05/17/2016 07:32 AM - matz (Yukihiro Matsumoto)

I vote for Thread#report_on_exception

Matz.

#40 - 05/17/2016 09:18 AM - Eregon (Benoit Daloze)

Akira Tanaka wrote:

> How about introducing Thread[.#]report_on_exception=(bool) ?
>
> false by default for compatibility.
> message should be printed at thread exit (not GC).

Agreed, except that it should be true by default otherwise it misses the whole point of Ruby Threads dying silently by default!
Extra output is very rarely a big problem and can be easily solved:

```
Thread.current.report_on_exception = false if Thread.current.respond_to? :report_on_exception
```

Not so nice, so maybe we could use keyword args on Thread.new as Charles suggested.
This could conflict with existing arguments to Thread.new but it seems mostly harmless
(it would just yield an extra argument to the block which would just get ignored on older versions).

But more importantly, it seems to always be wrong to ignore exceptions like that,
and Threads which can allow such exceptions should handle them explicitly by adding a begin/rescue/end around the whole body so they can perform a useful recovery strategy.

Ruby Threads are not "futures", they are heavy concurrent OS threads and therefore I believe the current behavior (dying silently) is counter productive for everyone.

#41 - 05/17/2016 12:01 PM - shyouhei (Shyouhei Urabe)

Benoit, I advise you to compromise on the default value. Once this feature gets implemented, its default value could be flipped later I think. We technically can't do what is proposed here now, which is definitely worse than a wrong default. no?

#42 - 05/17/2016 12:27 PM - akr (Akira Tanaka)

Benoit Daloze wrote:

> Agreed, except that it should be true by default otherwise it misses the whole point of Ruby Threads dying silently by default!

I have sympathy with you.
But it seems that it is difficult to persuade matz now.

However the class method, Thread.report_on_exception = true, is also accepted by matz.
This can be used to change the default.

I think that it is possible to change the default value
if we can show it doesn't cause problems.
Thread.report_on_exception = true can provide a way to experiment.

#43 - 05/17/2016 02:50 PM - nobu (Nobuyoshi Nakada)

Will messages be printed on all Threads if Thread.report_on_exception is true?
Or it is just the default value for each Threads at starting?

And, the initial value of Thread#report_on_exception is always false,
the value of the parent thread, or Thread.report_on_exception?

#44 - 05/17/2016 02:58 PM - dsferreira (Daniel Ferreira)

Why do we need Thread#report_on_exception ?
As I see it Thread.report_on_exception should be enough.

Is there a use case scenario for Thread#report_on_exception ?

The instance method brings added complexity that maybe we don't need to worry about.

#45 - 05/17/2016 03:26 PM - akr (Akira Tanaka)

Nobuyoshi Nakada wrote:

> Will messages be printed on all Threads if Thread.report_on_exception is true?
> Or it is just the default value for each Threads at starting?
>
> And, the initial value of Thread#report_on_exception is always false,
> the value of the parent thread, or Thread.report_on_exception?

I feel that the initial value of Thread#report_on_exception should be
Thread.report_on_exception.

The message is printed if a thread is exited with exception and
Thread#report_exception of the thread is true.

This means that
we can enable the message for each thread when Thread.report_on_exception = false and
we can disable the message for each thread when Thread.report_on_exception = true.

#46 - 05/17/2016 03:37 PM - nobu (Nobuyoshi Nakada)

Akira Tanaka wrote:

> This means that
> we can enable the message for each thread when Thread.report_on_exception = false and
> we can disable the message for each thread when Thread.report_on_exception = true.

Do you mean that setting it at the start up time, before starting other threads?
Or should the change affect the threads started before it?

#47 - 05/17/2016 03:41 PM - akr (Akira Tanaka)

Nobuyoshi Nakada wrote:

> Do you mean that setting it at the start up time, before starting other threads?

> Or should the change affect the threads started before it?

I expect that an assignment to Thread.report_on_exception doesn't affect
threads started before the assignment.

#48 - 05/17/2016 04:03 PM - spatulasnout (B Kelly)

Daniel Ferreira wrote:

> Why do we need Thread#report_on_exception ?
> As I see it Thread.report_on_exception should be enough.
>
> Is there a use case scenario for Thread#report_on_exception ?

Yes, because Thread#value passes the exception to the caller.

Granted ruby recently no longer supports $SAFE=4 sandboxing,
the following was a common sandbox idiom:

result = Thread.new {$SAFE=4; do_some_sandbox_work()}.value

In such cases, it is intended by design that any exception be
passed out of the sandbox to the caller. Automatic logging
at that boundary would not be desired.

By the way, I'm personally fine with logging by default, and
I support this feature as way to catch unexpected unhandled
thread exceptions.

But I would argue there indeed needs to be a way to disable
logging on a per-thread basis, where exceptions are intended
to be passed via Thread#value by design.

Regards,

Bill

#49 - 05/17/2016 05:33 PM - nobu (Nobuyoshi Nakada)

Akira Tanaka wrote:

> I expect that an assignment to Thread.report_on_exception doesn't affect
> threads started before the assignment.

Ok.

https://github.com/ruby/ruby/compare/trunk...nobu:feature/6647-report_on_exception

#50 - 05/17/2016 08:27 PM - headius (Charles Nutter)

> Benoit, I advise you to compromise on the default value. Once this feature gets implemented, its default value could be flipped later I think. We technically can't do what is proposed here now, which is definitely worse than a wrong default. no?

I am confused what it is we "can't" do right now. Do you mean: there's no reporting currently and we're seeking to add it...off by default but at least better than we have now?

I guess it's a *little* better because you could turn it on to see what's happening to all those badly-behaved threads, but it seems like having it off by default means most people never see any benefit.

I would try to keep convincing you all, but it seems the majority do not want it on by default. I'll just reiterate that I believe most of the purpose of this PR is lost if threads continue to die silently by default.

What about having verbose mode set Thread.report_on_exception = true? At least then people could pass a command-line flag to look for exception-killing threads without using -d and logging *all* exceptions.

1. User runs app with threads...threads disappear without reporting.
2. User runs app with -v (or some other flag to enable thread exception logging) to see why threads are dying.
3. User fixes app to properly handle exceptions in those threads.

#51 - 05/17/2016 11:14 PM - shyouhei (Shyouhei Urabe)

Charles Nutter wrote:

> Benoit, I advise you to compromise on the default value. Once this feature gets implemented, its default value could be flipped later I think. We technically can't do what is proposed here now, which is definitely worse than a wrong default. no?

I am confused what it is we "can't" do right now. Do you mean: there's no reporting currently and we're seeking to add it...off by default but at least better than we have now?

Yes. Sorry for my bad English. I wanted to say this is "better than nothing". I understand people want it with default on, and myself can live with that. However sticking to "default true, or no such feature" tactics can make this issue hard to land. Matz is not comfortable with this for years. We have just partially persuaded him about the functionality. I think this is a step forward.

#52 - 05/18/2016 02:32 AM - akr (Akira Tanaka)

I tried test-all with Thread.report_on_exception = true using nobu's patch.
https://patch-diff.githubusercontent.com/raw/ruby/ruby/pull/1357.patch

But most of reports are about joined threads.
They are not actual problem.

Now, I think report-on-exit (not on GC) should be disabled by default.

I recommend people wanting reports enabled by default should try similar test
with own applications.

I think reports enabled by default should be only (or mostly) for actual problems.
report-on-GC may be considerable, I think.

#53 - 05/18/2016 10:20 AM - Eregon (Benoit Daloze)

Akira Tanaka wrote:

> I tried test-all with Thread.report_on_exception = true using nobu's patch.
> https://patch-diff.githubusercontent.com/raw/ruby/ruby/pull/1357.patch
>
> But most of reports are about joined threads.
> They are not actual problem.

I also ran with nobu's patch and here is the result on my machine:
https://gist.github.com/eregon/811c8db1b91fac627b444ddc4c0f2760

What about DRb and these "can't alloc thread" for instance? It does not seem to be harmless.

There are of course tests using the fact exceptions propagate to #join or #value, but that is very coarse grained.
For example take TestQueue#test_deny_pushers, it's unclear whether pop or push throws the exception in
Thread.new{ synq.pop; q << i }, and the test would be more accurate by rescuing just the right error around the push:

```
thr = Thread.new{
  synq.pop
  q << i rescue $!
}
...
assert_kind_of ClosedQueueError, thr.value
```

Can we have a branch where the thread exceptions printed here are fixed?
Then I think we can potentially make a better judgment of whether it should be the default.
We might find a couple bugs along the way.

#54 - 05/18/2016 08:49 PM - headius (Charles Nutter)

> I wanted to say this is "better than nothing". I understand people want it with default on, and myself can live with that. However sticking to "default true, or no such feature" tactics can make this issue hard to land.

I agree. Whether it is on or off by default, I want to see the feature land. I'd like to see a way to enable it at command line (via -v or similar).

> Can we have a branch where the thread exceptions printed here are fixed?
> Then I think we can potentially make a better judgment of whether it should be the default.
> We might find a couple bugs along the way.

I agree. I look at this output and my first thought is not that we should silently swallow exceptions...it's that we have a lot of buggy code in stdlib that's letting threads die without handling the error properly.

Some examples:

Perhaps this isn't supposed to be a NoMethodError in the child?

```
 3) Failure:
TestDigest::TestDigestParen#test_race_mixed [/home/eregon/code/ruby/test/digest/test_digest.rb:257]:
assert_separately failed with error message
pid 29378 exit 0
|
| Thread terminated with
| -:9:in `block (2 levels) in <main>': undefined method `new' for nil:NilClass (NoMethodError)
```

This one comes up hundreds of times for all the threads it spins up. It's explicitly testing that a certain exception is thrown for each thread. Being able to do Thread.new(report: false) would solve all cases like this.

```
Thread terminated with
/home/eregon/code/ruby/test/thread/test_queue.rb:420:in `push': queue closed (ClosedQueueError)
 from /home/eregon/code/ruby/test/thread/test_queue.rb:420:in `block (4 levels) in test_deny_pushers'
```

Similar case here would be solved by report: false. This only comes up because it's testing Thread.abort_on_exception=, so it's an explicit case where it would not want reporting on.

```
  5) Failure:
TestThread#test_abort_on_exception [/home/eregon/code/ruby/test/ruby/test_thread.rb:297]:

1. [2/2] Assertion for "stderr"
   | <[]> expected but was
   | <["", "Thread terminated with", "-:3:in `block in <main>': unhandled exception"]>.
```

In every case I looked at, the warning is telling us something we can improve. Either it's an unexpected error getting swallowed and possibly breaking the test, or it's explicitly testing exceptions bubbling out of Thread.value. These are not cases I would want to see in my code.

#55 - 05/27/2016 08:44 AM - ko1 (Koichi Sasada)

any conclusion?

Another idea:

(1) Thread#report_on_exception = true
Show exception and backlog immediately (already proposed)

(2) Thread#report_on_exception = false (default)
Show exception and backlog at GC timing if exception is not handled.

I agree (1) is preferable, but breaks compatibility.
I agree #54, if people specify all of threads (which are joined after) report_on_exception = false (or specify report: false) will solve this compatibility.
But I'm not sure we can accept this approach.

(2) is not so good because the report will be delayed.
But not so bad compare with nothing to show.

#56 - 05/28/2016 01:08 AM - headius (Charles Nutter)

> (1) Thread#report_on_exception = true
> Show exception and backlog immediately (already proposed)
>
> (2) Thread#report_on_exception = false (default)
> Show exception and backlog at GC timing if exception is not handled.

I'm kinda coming around on the GC mechanism, if it's not going to be possible to report synchronously AND have this feature on by default.

I assume GC logging would only happen for threads that are not joined and on which value has not been called. So basically, by default any threads that you start up and walk away from will report that they've been terminated by an exception.

We will still have a few cases out there where the GC logging is unwanted, so I think there needs to be a way to turn it *completely* off. We have three states for VERBOSE and DEBUG, perhaps that works here?

- Thread#report_on_exception = true -- immediately report when the thread terminates due to an exception

- Thread#report_on_exception = nil (default) -- report when the thread object GCs without anyone looking at the exception
- Thread#report_on_exception = false -- no reporting of exception-triggered thread death for this thread

And I assume we would have Thread::report_on_exception{=} as well, right?

#57 - 05/28/2016 03:44 AM - headius (Charles Nutter)

Here's an implementation in JRuby: https://github.com/jruby/jruby/pull/3937

From the primary commit:

Implement Thread{.,#}report_on_exception[=].

This impl works as follows:

- Default global value is nil.
- nil = report when thread is GC if nobody has captured the exception (i.e. called #join or #value or abort_on_exception logic has fired).
- true = report when the thread terminates, regardless of capture.
- false = never report.
- New threads inherit the current global setting.
- If a thread name has been set from Ruby, it will be combined with JRuby's internal name for the report. If it has not been set, we will just report the internal thread name.

There are some open questions for this feature:

- If join/value are interrupted, should they still set the capture bit? My impl does not; they must complete.
- Is the VERBOSE-like nil/true/false clear enough or should we use symbols like :off, :gc, :on?

#58 - 05/30/2016 01:32 PM - Eregon (Benoit Daloze)

Koichi Sasada wrote:

> any conclusion?
>
> (2) Thread#report_on_exception = false (default)
> Show exception and backlog at GC timing if exception is not handled.

I am against reporting at GC time because:

- it might be surprising for the user to have exceptions shown at random times, possibly long after the thread dies
- it still shows nothing if the thread is still referenced (very easy, storing it in a local variable, in a constant, leaking by the VM, etc) or there is no GC
- in case of a thread dying and causing an application deadlock (so GC is unlikely), there is still no clue to the programmer and user of what happened
  (without -d which modifies the application behavior in a much larger way).

#59 - 06/06/2016 12:25 AM - nobu (Nobuyoshi Nakada)

- Status changed from Assigned to Closed

Applied in changeset r55290.

Thread.report_on_exception

- thread.c (thread_start_func_2): report raised exception if report_on_exception flag is set. [Feature #6647]

#60 - 06/13/2016 08:14 AM - akr (Akira Tanaka)

Now we can try Thread[.#]report_on_exception=(bool) using trunk.

It seems true-by-default and report-on-GC needs more discussion.

Anyway we can implement them later if some consensus met:
We can change the default value of Thread.report_on_exception and
we can implement Thread[.#]report_on_exception = :GC,
for example.

#61 - 07/18/2016 06:27 AM - backus (John Backus)

Akira Tanaka wrote:

> It seems true-by-default and report-on-GC needs more discussion.

I'm not sure how helpful my input is here but I think true-by-default makes this change useful. Otherwise it is probably not very useful. Right now it seems like the main problems are:

1. New ruby developers don't know to about Thread.abort_on_exception and end up confused when their code doesn't work but they aren't seeing exceptions
2. More experienced ruby developers can still easily forget to set Thread.abort_on_exception and be confused until they realize

These problems are not solved at all if the default value for Thread.report_on_exception is false:

1. New ruby developers wont know to set this configuration
2. Experienced ruby developers are probably going to forget report_on_exception if they forgot about abort_on_exception.

#62 - 09/13/2016 07:36 AM - djellemah (John Anderson)

Responding to 2.4.0-preview2 announcement on ruby-core. I have the opinion that a good default for Thread.report_on_exception is $VERBOSE, with this mapping:

nil - no reporting

false (-W1) - report exception on thread GC which is likely to generate fewer warnings. Which will be less verbose.

true (-W2 -w -v) - report exception on thread termination which is likely to generate extraneous warnings because threads haven't been joined yet. Which will be more verbose.

#63 - 10/11/2016 10:37 AM - shyouhei (Shyouhei Urabe)

We looked at this issue at developer meeting today and John's proposal sounded reasonable. So there quite are possibilities to accept it I think.

Problem is however, we currently do not implement "report exception on thread GC"-mode yet.

#64 - 12/06/2016 01:04 AM - shyouhei (Shyouhei Urabe)

- Related to Feature #13006: backtrace of thread killer added

#65 - 12/15/2016 10:05 PM - backus (John Backus)

Shyouhei Urabe wrote:

> We looked at this issue at developer meeting today and John's proposal sounded reasonable. So there quite are possibilities to accept it I think.

Fantastic! Any chance this will make it into the 2.4 release?

#66 - 10/15/2017 08:02 PM - Eregon (Benoit Daloze)

FWIW, I enabled Thread.report_on_exception = true by default in ruby/spec.
I had to change a few specs, but I think it really improved the specs rather than being cumbersome.
See https://github.com/ruby/spec/pull/517 for details.

I still believe we should have Thread.report_on_exception = true by default.
It only adds some extra stderr output for apps which let threads die, which is rarely intended anyway.
If it is intended, then one can use Thread.current.report_on_exception = false
to clarify it's OK for that thread to die and the failure is handled by the app on Thread#join.

#67 - 10/22/2017 01:53 AM - akr (Akira Tanaka)

Eregon (Benoit Daloze) wrote:

> FWIW, I enabled Thread.report_on_exception = true by default in ruby/spec.
> I had to change a few specs, but I think it really improved the specs rather than being cumbersome.
> See https://github.com/ruby/spec/pull/517 for details.

It seems we can reduce "Thread.current.report_on_exception = false"

if we implement report-on-GC because
some of "Thread.current.report_on_exception = false" is used
for threads to be joined.
So, I think we should try report-on-GC.
(it is not implemented, though)

#68 - 10/28/2017 12:09 PM - Eregon (Benoit Daloze)

Eregon (Benoit Daloze) wrote:

> FWIW, I enabled Thread.report_on_exception = true by default in ruby/spec.
> I had to change a few specs, but I think it really improved the specs rather than being cumbersome.
> See https://github.com/ruby/spec/pull/517 for details.

And it already proved useful!
https://travis-ci.org/ruby/spec/jobs/294063354
No way to know why we get ECONNREFUSED without seeing the other thread dying with EBADF.

akr:

> So, I think we should try report-on-GC.

I don't think report-on-GC is good, it just shows the error too late, if it shows it at all.
This is adding extra non-determinism on top of non-deterministic thread bugs, so I'm against it.

I argue that 99% of rubyists expect their threads to not die, or they plan for it and use Thread.new{begin ... rescue ... end} to log/handle exceptions.
Who wants a part of the program crashing silently? Nobody.

Thread#join is often too late, and it is a tiny price to pay to use Thread.report_on_exception = false in the rare case of using short-lived threads with no exception handling except #join.
Long-running threads are typically only joined at the end of the program, but the error, the fact that the thread died, should be shown to the programmer much earlier.

#69 - 10/29/2017 12:08 AM - rrroybbbean (RRRoy BBBean)

Maybe I don't understand this issue in sufficient detail, but...

I routinely use Thread.abort_on_exception=true when I want an
exception in a worker thread to raise an error in the main thread of
execution. If I don't specify this, then the worker thread fails
silently. This is predictable behavior.

If the default (predictable) behavior were to change, then how much
legacy code would be affected? How much stuff would break?

#70 - 10/29/2017 01:56 PM - Eregon (Benoit Daloze)

rrroybbbean (RRRoy BBBean) wrote:

> If the default (predictable) behavior were to change, then how much
> legacy code would be affected? How much stuff would break?

Having Thread.abort_on_exception=true would be too incompatible, I agree.

However, Thread.report_on_exception = true by default seems mostly compatible,
it's just extra stderr output when a thread dies from an exception, often revealing bugs in the code.

Warnings also cause extra stderr output, and we are not afraid to add those.
We could consider this feature a sort of warning which warns about threads dying (most likely) unexpectedly.

Currently, it seems report_on_exception does not use the warning system (i.e. Warning.warn), but I see no reason why it could not.
Then report_on_exception would be enabled on -W1 (default) and -W2, but silent on -W0.

Should we make Thread.report_on_exception use the warning system and have it behave like other rb_warn()?

Dear matz, could you give your opinion?

#71 - 11/19/2017 03:24 PM - Eregon (Benoit Daloze)

As an example, the bug in the test for https://bugs.ruby-lang.org/issues/13526#note-14
could have been detected much faster if Thread.report_on_exception was true by default.
report-on-gc would be useless here since all threads are held in the "threads" variable.

#72 - 11/29/2017 07:08 PM - Eregon (Benoit Daloze)

This issue is closed since the feature was implemented.
I'll continue the discussion of the default value in #14143.