Ruby - Bug #18810

Make `Kernel#p` interruptable.

05/28/2022 03:04 AM - ioquatix (Samuel Williams)

Status:	Closed		
Priority:	Normal		
Assignee:	ioquatix (Samuel Williams)		
Target version:			
ruby -v:		**Backport:**	2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: UNKNOWN

Description

While figuring out https://bugs.ruby-lang.org/issues/18465 I found a test which fails when rb_io_flush becomes blocking.: https://github.com/ruby/ruby/commit/fe6b2e20e9f17ed2c2900aa72994e075ffdc7124

It seems unusual to me that Kernel#p is uninterruptible (unique among all Ruby methods). I'd like to make Kernel#p interruptible.

Associated revisions

Revision ae609a995e344877a990f4c16eca88b02dab5eba - 08/30/2023 08:21 PM - jeremyevans (Jeremy Evans)

Document that Kernel#p is for debugging and may be uninterruptible [ci skip]

Fixes [Bug #18810]

Revision ae609a995e344877a990f4c16eca88b02dab5eba - 08/30/2023 08:21 PM - jeremyevans (Jeremy Evans)

Document that Kernel#p is for debugging and may be uninterruptible [ci skip]

Fixes [Bug #18810]

Revision ae609a99 - 08/30/2023 08:21 PM - jeremyevans (Jeremy Evans)

Document that Kernel#p is for debugging and may be uninterruptible [ci skip]

Fixes [Bug #18810]

History

#1 - 05/28/2022 04:16 AM - ioquatix (Samuel Williams)

PR: https://github.com/ruby/ruby/pull/5967

#2 - 05/30/2022 12:44 AM - mame (Yusuke Endoh)

Since Kernel#p is a method for debugging, I think this spec would be useful. If it is made interruptable, it will be difficult to use Kernel#p in a block of Thread.handle_interrupt(TimeoutError => :on_blocking).

#3 - 07/21/2022 12:00 PM - mame (Yusuke Endoh)

We discussed this issue at the dev meeting. We will add a document to Kernel#p so that it is uninterruptible and for debugging purpose.

#4 - 07/22/2022 05:51 AM - ioquatix (Samuel Williams)

@mame (Yusuke Endoh) I understand the discussion and I'm okay with the outcome, but I still don't understand why being uninterruptible matters in practice. I'm still a little concerned this can hang the interpreter, but I don't know for sure - because remember the internal write function can call the fiber scheduler. So it can potentially execute a lot of Ruby code in an uninterruptible way which is very unique - it is the only method which behaves like this.

#5 - 07/22/2022 09:56 AM - Eregon (Benoit Daloze)

mame (Yusuke Endoh) wrote in #note-2:

> Since Kernel#p is a method for debugging, I think this spec would be useful. If it is made interruptable, it will be difficult to use Kernel#p in a block of Thread.handle_interrupt(TimeoutError => :on_blocking).

You mean if p is used inside a Timeout.timeout block?
And you'd worry the object would be printed but no newline, or the write would be interrupted in the middle and have written some partial output (is that even possible when writing to a terminal given write is atomic in at least that case?)?

If it's about the newline, that would be solved by p appending "\n" to the string and only doing a single write, everything is a million times simpler that way.

I think this is a very rare case and any developer must expect this or far worse effects of asynchronous exceptions when using Timeout.timeout/Thread#raise.

> We discussed this issue at the dev meeting. We will add a document to Kernel#p so that it is uninterruptible and for debugging purpose.

IMHO this should not be part of specification, but an implementation choice/detail.
So I'd suggest to write it like "Kernel#p is uninterruptible on CRuby to avoid being interrupted by Timeout.timeout" or so in the docs.

#6 - 07/26/2022 08:34 AM - mame (Yusuke Endoh)

My concern is that inserting p(...) changes a program behavior unintentionally (except that the p writes something to stdout, of course).

When debugging, we often insert p(...) and run the program to reproduce the bug being debugged. It would be annoying if the bug becomes unreproducible because the insertion of p(...) changes the behavior of the program. I don't want to see something like the "bug not reproduced on the debugger" problem.

#7 - 07/26/2022 10:14 PM - ioquatix (Samuel Williams)

> My concern is that inserting p(...) changes a program behavior unintentionally (except that the p writes something to stdout, of course).

There are so many ways it can do this. If the fiber scheduler is active it is a totally different code path. What about bufferred output (common on stdout).

What you really want is a blocking write to stderr, e.g. fprintf(stderr, ...). This has very few side effects.

#8 - 07/30/2022 11:38 AM - Eregon (Benoit Daloze)

@mame (Yusuke Endoh) Could you give an example in Ruby code?
I don't understand your concern, of course p changes behavior because it calls inspect (which can do anything) and prints, but that seems fully expected.

#9 - 06/22/2023 11:27 PM - jeremyevans0 (Jeremy Evans)

I submitted a pull request to update the Kernel#p documentation: https://github.com/ruby/ruby/pull/7975

#10 - 06/23/2023 10:24 AM - Eregon (Benoit Daloze)

p being uninterruptible means all of object.inspect and potential Fiber scheduler code is run under Thread.handle_interrupt(Object => :never) { ... }.
That seems a really bad idea, because e.g. if inspect would take a long time (creates a huge String or buggy), then it's not possible to interrupt it, e.g. with Ctrl+C.
I think this is a CRuby bug, it's too dangerous to disable interrupts for arbitrary code.

#11 - 06/23/2023 10:26 AM - Eregon (Benoit Daloze)

Mmh, but it does work to interrupt p:

```
$ ruby -e 'o=Object.new; def o.inspect; loop { Thread.pass }; end; p 1; p o'
1
^C-e:1:in `pass': Interrupt
 from -e:1:in `block in inspect'
 from -e:1:in `loop'
 from -e:1:in `inspect'
 from -e:1:in `p'
 from -e:1:in `<main>'
```

What am I missing then?

#12 - 06/23/2023 10:34 AM - Eregon (Benoit Daloze)

It's because the code changed on master, https://github.com/ruby/ruby/commit/fe6b2e20e9f17ed2c2900aa72994e075ffdc7124 had that bug but master doesn't call inspect under rb_uninterruptible().
On master:

```
static VALUE
rb_f_p(int argc, VALUE *argv, VALUE self)
{
    int i;
    for (i=0; i<argc; i++) {
```

```
        VALUE inspected = rb_obj_as_string(rb_inspect(argv[i]));
        rb_uninterruptible(rb_p_write, inspected);
    }
    return rb_p_result(argc, argv);
}

static VALUE
rb_p_result(int argc, const VALUE *argv)
{
    VALUE ret = Qnil;

    if (argc == 1) {
        ret = argv[0];
    }
    else if (argc > 1) {
        ret = rb_ary_new4(argc, argv);
    }
    VALUE r_stdout = rb_ractor_stdout();
    if (RB_TYPE_P(r_stdout, T_FILE)) {
        rb_uninterruptible(rb_io_flush, r_stdout);
    }
    return ret;
}
```

So only the write and the flush are uninterruptible now.
So it does not seem correct to say Kernel#p is uninterruptible.

These rb_uninterruptible() seem to have very little effect, indeed it's only if there was a Thread.handle_interrupt(TimeoutError => :on_blocking) around the p and there is an interrupt and STDOUT is not just a TTY or a file where write cannot be interrupted anyway IIRC, then it could happen the interrupt is done during p's write (or flush), instead of another blocking call in that block.
I think that's fully expected, we are trying to tweak a corner of a very rare case by making things complicated, it seems not worth it.

#13 - 08/30/2023 08:21 PM - jeremyevans (Jeremy Evans)

- Status changed from Open to Closed

Applied in changeset git|ae609a995e344877a990f4c16eca88b02dab5eba.

Document that Kernel#p is for debugging and may be uninterruptible [ci skip]

Fixes [Bug #18810]