Ruby master - Bug #18810
Make `Kernel#p` interruptable.
05/28/2022 03:04 AM - ioquatix (Samuel Williams)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Open</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>ioquatix (Samuel Williams)</td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
</tr>
<tr>
<td>ruby -v:</td>
<td></td>
</tr>
<tr>
<td>Backport:</td>
<td>2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: UNKNOWN</td>
</tr>
</tbody>
</table>

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/fe6b2e20e9f17ed2c2900aa72994e075f7dc7124

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

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.
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 buffered output (common on stdout).

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

@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.