Ruby master - Feature #18411

Introduce `Fiber.blocking` for disabling scheduler.

12/16/2021 07:58 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></td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

When implementing pure-ruby IO scheduler, we may need to invoke some Ruby IO operations without entering the scheduler.

```ruby
def io_write(fiber, io, buffer, length)
  offset = 0

  while length > 0
    # From offset until the end:
    chunk = buffer.to_str(offset, length)
    case result = io.write_nonblock(chunk, exception: false)
      when :wait_readable
        self.io_wait(fiber, io, IO::READABLE)
      when :wait_writable
        self.io_wait(fiber, io, IO::WRITABLE)
      else
        offset += result
        length -= result
      end
    end

  return offset
end
```

There are some cases where even in this code read_nonblock can invoke fiber scheduler creating infinite recursion.

Therefore, I propose to introduce Fiber.blocking{...} which has almost identical implementation to Fiber.new(blocking: true) {}.resume.

In the above code, we change the line:

```ruby
  case result = io.write_nonblock(chunk, exception: false)
```

To

```ruby
  case result = Fiber.blocking{io.write_nonblock(chunk, exception: false)}
```

This ensures that write_nonblock can never enter the scheduler again.

**History**

#1 - 12/16/2021 08:00 AM - ioquatix (Samuel Williams)

matz (Yukihiro Matsumoto) this isn't strictly necessary but makes pure Ruby implementation of IO read/write hooks more efficient. I'm not sure if there is valid case for general user code. Can we consider it for Ruby 3.1?

#2 - 12/18/2021 02:52 AM - ioquatix (Samuel Williams)

- Description updated

#3 - 12/23/2021 05:10 PM - jeremyevans0 (Jeremy Evans)

- Backport deleted (2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN)
- Tracker changed from Bug to Feature