Ruby - Misc #19122

Use MADV_DONTNEED instead of MADV_FREE when freeing a Fiber's stack

11/11/2022 04:06 PM - smcgivern (Sean McGivern)

Status:	Assigned	
Priority:	Normal	
Assignee:	ioquatix (Samuel Williams)	

Description

I'd like to propose that Ruby stops using MADV_FREE when freeing a Fiber's stack, and switches to using MADV_DONTNEED even when MADV_FREE is supported.

MADV_FREE is used in one place in the Ruby codebase, when freeing the stack of a freed Fiber: https://git.ruby-lang.org/ruby.git/tree/cont.c#n683

The comment for fiber_pool_stack_free says:

```
// We advise the operating system that the stack memory pages are no longer being used.
// This introduce some performance overhead but allows system to relaim memory when there is press
ure.
```

Where possible (i.e. on Linux 4.5 and later), fiber_pool_stack_free uses MADV_FREE over MADV_DONTNEED. This has the side effect that memory statistics such as RSS will not reduce until and unless the OS actually reclaims that memory. If that doesn't happen, then the reported memory usage via RSS will be much higher than the 'real' memory usage.

If this was pervasive throughtout the Ruby codebase then that would be one thing, but currently this is just for Fiber. This means that:

1. A program that doesn't use Fiber will have somewhat reliable RSS statistics on recent Linux.
2. A program that heavily uses Fiber (such as something using Async::HTTP) will see an inflated RSS statistic.

Go made a similar change to the one I'm proposing here for similar reasons: https://github.com/golang/go/issues/42330

> While MADV_FREE is somewhat faster than MADV_DONTNEED, it doesn't affect many of the statistics that MADV_DONTNEED does until the memory is actually reclaimed. This generally leads to poor user experience, like confusing stats in top and other monitoring tools; and bad integration with management systems that respond to memory usage.
> [...]
> I propose we change the default to prefer MADV_DONTNEED over MADV_FREE, to favor user-friendliness and minimal surprise over performance. I think it's become clear that Linux's implementation of MADV_FREE ultimately doesn't meet our needs.

As an aside, MADV_FREE was not used in Ruby 3.1 (https://bugs.ruby-lang.org/issues/19101), and I haven't found any bugs filed about this behaviour other than that one.

History

#1 - 11/11/2022 04:06 PM - smcgivern (Sean McGivern)

@ioquatix (Samuel Williams) apologies for the direct assignment; you just seemed like the person who has the most knowledge and investment in the current state of Fiber.

#2 - 11/21/2022 04:18 AM - ioquatix (Samuel Williams)

I don't have a strong opinion about this, but I'm generally against loosing performance.

Maybe it can be controlled using environment variable.

#3 - 11/25/2022 10:06 AM - smcgivern (Sean McGivern)

ioquatix (Samuel Williams) wrote in #note-2:

> I don't have a strong opinion about this, but I'm generally against loosing performance.
>
> Maybe it can be controlled using environment variable.

Having this user-controllable works for me, although I don't know about the Ruby project's general stance on toggles like this.

@ioquatix (Samuel Williams) is https://github.com/socketry/falcon-benchmark the right test suite to run here? I'm not sure if that's what you ran in https://bugs.ruby-lang.org/issues/15997#note-19, or if that was slightly different.

I'm happy to try this out with the various options here to quantify the current state - just point me in the right direction :-)

#4 - 05/24/2023 03:08 PM - ioquatix (Samuel Williams)

https://github.com/ruby/ruby/pull/7855 should enable you to test different advice. Just bit shift it to the left to enable any MADV_... value to be specified.

#5 - 05/25/2023 02:07 AM - ioquatix (Samuel Williams)

If you want to use a specific mode (OS specific), you can do this:

On Linux, find the mode, e.g. MADV_DONTNEED = 6 (https://github.com/torvalds/linux/blob/933174ae28ba72ab8de5b35cb7c98fc211235096/arch/alpha/include/uapi/asm/mman.h#L50).

Shift it one bit to the left (i.e. multiply by 2) = 12.

Then run Ruby like this:

```
> RUBY_SHARED_FIBER_POOL_FREE_STACKS=12 ruby
```

This will force the use of MADV_DONTNEED. However, no validation is done, so the OS may not accept it. So it will also emit a warning that this behaviour is OS dependent and may crash your Ruby interpreter.

With this in place, you should be able to test your code/hypothesis/memory usage/performance.

As for making this the default, I suppose we could consider it but we'd need to confirm the performance impact. I would prefer to have high performance by default. RSS is a poor measurement IMHO. What about USS and PSS? Are they impacted the same way? Can we subtract the usage that can be later dropped?

#6 - 05/30/2023 12:27 PM - smcgivern (Sean McGivern)

ioquatix (Samuel Williams) wrote in #note-5:

> If you want to use a specific mode (OS specific), you can do this:
>
> On Linux, find the mode, e.g. MADV_DONTNEED = 6 (https://github.com/torvalds/linux/blob/933174ae28ba72ab8de5b35cb7c98fc211235096/arch/alpha/include/uapi/asm/mman.h#L50).
>
> Shift it one bit to the left (i.e. multiply by 2) = 12.
>
> Then run Ruby like this:
>
> ```
> > RUBY_SHARED_FIBER_POOL_FREE_STACKS=12 ruby
> ```

Thanks for adding this!

> As for making this the default, I suppose we could consider it but we'd need to confirm the performance impact. I would prefer to have high performance by default. RSS is a poor measurement IMHO. What about USS and PSS? Are they impacted the same way? Can we subtract the usage that can be later dropped?

I mean, as I said in the original description, this is the only part of Ruby that works this way, which makes it surprising. If usage of fibers becomes more common in the wild, or MADV_FREE gets used in other areas where Ruby frees memory, then I would predict more reports noting this behaviour. Other languages have reverted their usage of MADV_FREE for similar reasons.

We can construct another metric that gives us exactly what we want (for cAdvisor, we'd need something like https://github.com/google/cadvisor/issues/3197), but not through PSS or USS; both of those track RSS closely here, at least in my tests. We need to look at either active_anon + inactive_anon (for a cgroups memory.stat file), or LazyFree plus some other things I'm not sure about (for a process's smaps_rollup). This won't be the default in most monitoring tools, or in top et al, so as I said above, that wouldn't necessarily stop issues coming in.

#7 - 04/03/2024 03:50 AM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Assigned

#8 - 03/04/2025 09:10 PM - ioquatix (Samuel Williams)

On the latest Linux kernel, the constants have changed:

```
MADV_DONTNEED = 4
MADV_FREE = 8
```

I did a quick comparison, creating 10,000 fibers x10 times in a loop:

```
samuel@aiko ~/D/s/memory-leak (main)> time RUBY_SHARED_FIBER_POOL_FREE_STACKS=8 bundle exec test.rb >/dev/null
<main>: warning: Setting RUBY_SHARED_FIBER_POOL_FREE_STACKS to a value greater than 1 is operating system spec
ific, and may cause crashes.

_____________________________________________________
Executed in  841.07 millis    fish           external
   usr time  373.88 millis  441.00 micros  373.44 millis
   sys time  463.22 millis  168.00 micros  463.05 millis

samuel@aiko ~/D/s/memory-leak (main)> time RUBY_SHARED_FIBER_POOL_FREE_STACKS=16 bundle exec test.rb >/dev/nul
l
<main>: warning: Setting RUBY_SHARED_FIBER_POOL_FREE_STACKS to a value greater than 1 is operating system spec
ific, and may cause crashes.

_____________________________________________________
Executed in  582.77 millis    fish           external
   usr time  357.66 millis  420.00 micros  357.24 millis
   sys time  220.90 millis  164.00 micros  220.74 millis

samuel@aiko ~/D/s/memory-leak (main)> time RUBY_SHARED_FIBER_POOL_FREE_STACKS=0 bundle exec test.rb >/dev/null

_____________________________________________________
Executed in  444.61 millis    fish           external
   usr time  312.92 millis  503.00 micros  312.41 millis
   sys time  128.99 millis    0.00 micros  128.99 millis
```

Also memory usage as we expected:

```
samuel@aiko ~/D/s/memory-leak (main)> RUBY_SHARED_FIBER_POOL_FREE_STACKS=16 bundle exec test.rb
<main>: warning: Setting RUBY_SHARED_FIBER_POOL_FREE_STACKS to a value greater than 1 is operating system spec
ific, and may cause crashes.
PID: 70235
Memory usage: 150.82 MB
Clearing fibers and garbage collecting...
Memory usage: 150.82 MB
samuel@aiko ~/D/s/memory-leak (main)> RUBY_SHARED_FIBER_POOL_FREE_STACKS=8 bundle exec test.rb
<main>: warning: Setting RUBY_SHARED_FIBER_POOL_FREE_STACKS to a value greater than 1 is operating system spec
ific, and may cause crashes.
PID: 70279
Memory usage: 151.63 MB
Clearing fibers and garbage collecting...
Memory usage: 73.63 MB
```

Using the following test script:

```ruby
#!/usr/bin/env ruby

require "memory/leak/system"

def print_memory_usage
  size = Memory::Leak::System.memory_usage(Process.pid)
  units = %w(B KB MB GB TB)
  unit = 0

  while size > 1024 && unit < units.size
    size /= 1024.0
    unit += 1
  end

  puts "Memory usage: %.2f %s" % [size, units[unit]]
end

puts "PID: #{Process.pid}"

10.times do |i|
  puts "Iteration: #{i}"

  # Create 10,000 fibers:
  fibers = 10_000.times.map do
    Fiber.new do
      Fiber.yield Fiber.current
    end.resume
  end
```

```
  # Print memory usage:
  print_memory_usage

  puts "Clearing fibers and garbage collecting..."

  # Clear fibers:
  fibers.clear
  GC.start

  # Print memory usage:
  print_memory_usage
end
```