

Ruby trunk - Bug #3924

Performance bug (in require?)

10/10/2010 08:15 AM - cabo (Carsten Bormann)

Status: Closed	
Priority: Normal	
Assignee: yugui (Yuki Sonoda)	
Target version: 2.0.0	
ruby -v: -	Backport:
Description =begin Running irb < /dev/null in 1.9.2 causes 3016 calls to lstat64. For instance, there is a sequence of 28 repetitions each of lstat calls to all 6 non-empty path prefixes of /opt/local/lib/ruby1.9/1.9.1/irb.rb -- a total of 170 lstats apparently just to load this file; another set of lstats then occurs later for another 18 (times 6) times. Clearly, something is running amok in the calling sequence rb_require_safe -> realpath_rec -> lstat. Another example: Running a simple test with the baretest gem causes 17008 calls to lstat. According to perftools.rb, 80 % of the 1.2 seconds of CPU is used in Kernel#gem_original_require (and another 12 in GC, some of which may be caused by this). =end	
Related issues:	
Related to Ruby trunk - Bug #7158: require is slow in its bookkeeping; can ma...	Closed 10/14/2012
Is duplicate of Ruby trunk - Feature #3906: Initializing and loading Rails en...	Closed 10/05/2010

History

#1 - 10/21/2010 05:47 PM - naruse (Yui NARUSE)

- Target version set to 1.9.3

=begin
As you know, it's a problem from gem's require.
It is planed that 1.9.3 fixes this.
=end

#2 - 10/21/2010 11:18 PM - luislavena (Luis Lavena)

=begin
Yui,

Are you sure about this? because even disabling RubyGems and removing everything from \$LOAD_PATH the IO operations performed by Ruby for a simple require are too much.

Ruby 1.8 also suffered from excessive IO operations even without RubyGems. I can provide Windows trace logs of these items if you want.
=end

#3 - 05/15/2011 03:39 PM - xaviershay (Xavier Shay)

Progress update time!

tl;dr - I've made the performance linear, still need to do a bit more clean up though.

I have started switching out the current loop of \$LOADED_FEATURES to use a hash look up (using st.h) in order to fix this problem. You can try it now by using require_2 rather than require on my fork (or by changing the mapping of require at the bottom of load.c):
<https://github.com/xaviershay/ruby/compare/require-performance-fix>

This fork passes all ruby tests and rubyspec require tests. It can also load a rails stack. That doesn't mean it is correct (the tests are not comprehensive), but it's a start.

It displays near-linear performance. Here are some graphs:
versus 1.9 - <https://skitch.com/xaviershay/r9pwb/tc-load-time>
versus 1.8 - <https://skitch.com/xaviershay/r9pwn/tc-load-time>

This is a pretty big patch, and also my first on MRI so there are a few potential issues:

- This is pretty much a rewrite for load.c, since I couldn't get my head around rb_feature_p. That means it's risky. It is likely possibly to add a

hash-table lookup to the existing architecture, but someone else would have to volunteer to do this. On the up side, it is far easier to follow the algorithm in the new code.

- It doesn't use the file search functions in file.c, so I suspect there could be some safe level issues not covered yet.
- I have made public one or two functions in file.c and array.c. Pretty sure I could rewrite what I have to avoid doing this.
- rb_provide no longer makes sense. It was being used in enumerator.c but I found another method of providing backwards compatibility. Since it is in intern.h, it can maybe just be removed?
- I use a proxy class for \$LOADED_FEATURES to keep the loaded features hash up to date. Bit of a hack but I couldn't come up with a nicer way of doing it.
- Is it OK to add a new member to the VM struct? I added new_loaded_features (to be renamed shortly).
- Not tested on windows yet.

Outstanding tasks still on my list (hopefully will be able to do in the next week):

- Autoload still uses the old rb_feature_provided.
- It is much faster on synthetic benchmarks, but still slower on many real world cases because I have only optimized the general algorithm so far and still am using a lot of rb_funcalls that are not necessary.
- I haven't marked all my methods as static yet.
- Remove old require code once I'm satisfied the new code works.
- Clean up the vm->new_loaded_features stuff.

Re the original ticket description, I saw the lstats in profiling too but believe it is a symptom not a cause.

#4 - 05/16/2011 07:31 AM - mgenereu (Michael Genereux)

Yes on the red herring of lstat. I removed the recursive lstat'ing (it's checking the path for symbolic links) and there was no speed increase.

#5 - 05/16/2011 11:38 AM - xaviershay (Xavier Shay)

Progress update:

Have done general cleanup to make the code more readable, including marking local methods as static. Optimized a lot of my code that was previously using rb_funcall, now it actually does things properly in C.

I have switched autoload over to the new system, but there is still a problem with it - ActiveRecord doesn't load properly due to a constant missing which should be autoloaded. Not sure what is causing this.

There is a failing test in rubyspec for autoload which I thought might be related, but it appears to be broken against ruby trunk also.

#6 - 05/20/2011 07:28 PM - xaviershay (Xavier Shay)

Progress update:

Fixed the autoload problem, my fork now loads a moderate sized rails app in the same time as ruby 1.9.2 (and far faster than 1.9.3dev 20s compared to 46s), as well as passing all other tests/specs. This proves the algorithm I think, the next step is to optimize the inner functions (such as rb_locate_file_with_extensions), since they still use plenty of rb_funcall to do string manipulation which could be far faster done in C. There is also room for improvement by caching rb_locate_file.

Code is still at <https://github.com/xaviershay/ruby/tree/require-performance-fix>

I hope to have more time to work on it on Sunday.

#7 - 05/22/2011 03:33 PM - xaviershay (Xavier Shay)

Progress update:

Didn't get much time to work on this today, but came up with a benchmark to measure performance with a full \$LOAD_PATH (suspected issue with slow rails loading), which indeed shows mine to have the same performance curve as 1.9.3, but with a higher multiplier: <https://gist.github.com/985224> (graph in the comments)

Doubling the number of available extensions (.bogus, .bogus2, etc...) doubled the multiplier, suggesting improvement in rb_locate_file_with_extensions could be a win.

#8 - 05/23/2011 07:56 PM - xaviershay (Xavier Shay)

- File require_performance.diff added

Progress update:

Didn't make much impact on the graph in prior update, but also may have been to eager to blame slow rails loading on it - rails only has ~50 entries in the load path.

I added a cache to file expansion (so it remembers the expanded path of "active_record" without having to search for it), which gave a ~0.5s speed up on a blank rails app, and ~6s on a medium sized one. This patch is now faster than 1.9.2 in all benchmarks I have, and way faster than 1.9.3dev (though it has been that for a while). I suspect there are further optimizations that could be made, especially now the code is easier to follow.

I have attached a diff. It passes make test, make test-all, and a few big rails test suites. There are still some outstanding issues on which need

discussion.

- 1) This is pretty much a rewrite for load.c, since I couldn't get my head around rb_feature_p. That means it's risky. It is likely possibly to add a hash-table lookup to the existing architecture, but someone else would have to volunteer to do this. On the up side, it is far easier to follow the algorithm in the new code.
- 2) It doesn't use the file search functions in file.c, so I suspect there could be some safe level issues not covered yet.
- 3) I have made public (non-static) one or two functions in file.c and array.c. Is this OK? Pretty sure I could rewrite to avoid doing this if necessary.
- 4) rb_provide doesn't really make sense. It was being used in enumerator.c but I found another method of providing backwards compatibility. Since it is in intern.h, it can maybe just be removed?
- 5) I use a proxy class for \$LOADED_FEATURES to keep the loaded features hash up to date. Is this OK? Bit of a hack but I couldn't come up with a nicer way of doing it.
- 6) Is it OK to add a new member to the VM struct? I added new_loaded_features (to be renamed shortly).
- 7) Not tested on windows yet.
- 8) The code would be nicer if some related functions could be moved to another file (rb_locate_*), I don't know the best way to do this though.

#9 - 05/23/2011 08:47 PM - xaviershay (Xavier Shay)

It appears this commit:

<https://github.com/xaviershay/ruby/commit/d7d6c41524ec8bcda8d6672e7d8b7ae812abc239>

breaks make clean && make with this error:

<https://gist.github.com/cb99899d0918a840ab76>

I have no idea why. Perhaps switching over require to the new code? But the compile is failing still in C land... Suggestions welcome.

#10 - 05/24/2011 10:29 AM - normalperson (Eric Wong)

I'll try to dedicate some time to helping on this but I can't promise anything. I'll take a longer look at Xavier's work later tonight or tomorrow.

However, I needed the following trivial patch to build with gcc 4.4.5 on Debian:

```
diff --git a/variable.c b/variable.c
index 551a7d5..908a649 100644
--- a/variable.c
+++ b/variable.c
@@ -19,6 +19,7 @@
#include "constant.h"
#include "internal.h"

+VALUE rb_feature_provided_2(VALUE); /* move to ruby/intern.h ? */
void rb_vm_change_state(void);
void rb_vm_inc_const_missing_count(void);
```

--
Eric Wong

#11 - 05/24/2011 04:42 PM - xaviershay (Xavier Shay)

I figured out how to make my branch compile (see my last comment), but I have no idea why I have to do this or why it works. It's a hack workaround, not a real fix. I can compile trunk no worries:

First few steps the same as normal

```
cd /path/to/source/directory
autoconf
cd /path/to/build/directory
/path/to/source/directory/configure
```

HACKS follow

The parenthesis aren't escaped in this file, which causes a bash syntax error.

```
sed -i " s/INIT_FUNCS(X)/INIT_FUNCS\\(X\\)/ ext/-test-/string/Makefile
```

-e picks up variables from my environment, top_srcdir isn't set properly otherwise

```
make -e top_srcdir=/path/to/source/directory
```

I have no experience with Makefiles/configure, can anyone suggest what might be causing this problem?

#12 - 05/28/2011 10:11 AM - xaviershay (Xavier Shay)

My last problem was related to the build system pushing things into "\$" and bypassing require.

I now have a patch that I would like to submit for consideration:

<https://gist.github.com/996418>

I am going to try and get some more eyeballs on this to test it more thoroughly:

<http://rhn.net/2011/05/28/speeding-up-rails-startup-time>

#13 - 05/28/2011 10:23 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Great work, Xavier!

I would like to try it but my RVM installation seems to be broken somehow:

```
rvm install ruby-head --patch /tmp/require-performance-fix.patch
```

Installing Ruby from source to:

/home/rodrigo/.rvm/rubies/ruby-1.8.7-p334, this may take a while depending on your cpu(s)...

I don't know why it insists on 1.8.7, but I'll try it manually as soon as I have some time, thanks!

I'll let you know the feedbacks.

Regards,

Rodrigo.

Em 27-05-2011 22:11, Xavier Shay escreveu:

Issue [#3924](#) has been updated by Xavier Shay.

My last problem was related to the build system pushing things into "\$" and bypassing require.

I now have a patch that I would like to submit for consideration:

<https://gist.github.com/996418>

I am going to try and get some more eyeballs on this to test it more thoroughly:

<http://rhn.net/2011/05/28/speeding-up-rails-startup-time>

Bug [#3924](#): Performance bug (in require?)

<http://redmine.ruby-lang.org/issues/3924>

Author: Carsten Bormann

Status: Open

Priority: Normal

Assignee:

Category: core

Target version: 1.9.3

ruby -v: ruby 1.9.2p0 (2010-08-18 revision 29036) [x86_64-darwin10]

```
=begin
```

```
Running irb< /dev/null in 1.9.2 causes 3016 calls to lstat64.
```

For instance, there is a sequence of 28 repetitions each of lstat calls to all 6 non-empty path prefixes of /opt/local/lib/ruby1.9/1.9.1/irb.rb -- a total of 170 lstats apparently just to load this file; another set of lstats then occurs later for another 18 (times 6) times. Clearly, something is running amok in the calling sequence rb_require_safe -> realpath_rec -> lstat.

Another example: Running a simple test with the baretest gem causes 17008 calls to lstat. According to perftools.rb, 80 % of the 1.2 seconds of CPU is used in Kernel#gem_original_require (and another 12 in GC, some of which may be caused by this).

```
=end
```

#14 - 05/29/2011 01:23 AM - luislavena (Luis Lavena)

- ruby -v changed from ruby 1.9.2p0 (2010-08-18 revision 29036) [x86_64-darwin10] to -

On Fri, May 27, 2011 at 9:11 PM, Xavier Shay xavier-list@rhn.net wrote:

My last problem was related to the build system pushing things into "\$" and bypassing require.

I now have a patch that I would like to submit for consideration:
<https://gist.github.com/996418>

Windows 7, x64, building with GCC 4.5.2 (32bits).

Patch applied to r31761, with the following warnings:

```
C:\Users\Luis\Projects\oss\ruby>git apply require-performance-fix-r31758.patch
require-performance-fix-r31758.patch:115: trailing whitespace.
/*
require-performance-fix-r31758.patch:128: trailing whitespace.
/*
require-performance-fix-r31758.patch:648: trailing whitespace.
basename
```

#15 - 05/29/2011 01:53 AM - jonforums (Jon Forums)

[luis \(Luis Lopez\)](#), if you have time, would you add results to your gist using a patched ruby_1_9_2 build?

Your 1.9.3dev results on mingw32 are amazing...it would be great to see if the Xavier's patch makes sense to backport before Yugui releases an updated 1.9.2

#16 - 05/29/2011 01:57 AM - luislavena (Luis Lavena)

Jon Forums wrote:

[luis \(Luis Lopez\)](#), if you have time, would you add results to your gist using a patched ruby_1_9_2 build?

Patch doesn't apply against 1.9.2:

```
C:\Users\Luis\Projects\oss\ruby>git apply require-performance-fix-r31758.patch
require-performance-fix-r31758.patch:115: trailing whitespace.
/*
require-performance-fix-r31758.patch:128: trailing whitespace.
/*
require-performance-fix-r31758.patch:648: trailing whitespace.
basename = rb_funcall(rb_cFile, rb_intern("basename"), 2,
require-performance-fix-r31758.patch:652: trailing whitespace.
file_name_with_extension = rb_funcall(rb_cFile, rb_intern("join"), 2,
require-performance-fix-r31758.patch:728: trailing whitespace.
* This is the most common case, so optimize for it. If not found, fall
warning: enc/make_encmake.rb has type 100644, expected 100755
error: patch failed: enc/make_encmake.rb:3
error: enc/make_encmake.rb: patch does not apply
warning: ext/extmk.rb has type 100644, expected 100755
error: patch failed: load.c:68
error: load.c: patch does not apply
error: patch failed: test/ruby/test_require.rb:339
error: test/ruby/test_require.rb: patch does not apply
error: patch failed: variable.c:19
error: variable.c: patch does not apply
error: patch failed: vm_core.h:324
error: vm_core.h: patch does not apply
```

#17 - 05/29/2011 06:37 AM - xaviershay (Xavier Shay)

Rodrigo: Please try the latest version of rvm: rvm get head. Otherwise let me know what platform you are on.

Luis: Sorry I'm not sure what versions I ran my benchmarks against. Poor form on my part, though there was a recent commit on May 23rd which I would imagine sped things up: r31712, 7d20942bf6dd54cf54d12d4c9f9dc86b9a813bb4

#18 - 05/29/2011 06:39 AM - xaviershay (Xavier Shay)

Actually Luis I just read your gist, my numbers match up with ruby 1.9.3dev (2011-05-28 trunk 31761) [i386-mingw32], I never saw it as bad as 52s for r31496.

#19 - 05/29/2011 09:18 PM - mame (Yusuke Endoh)

Hello, Xavier

Thank you for your contribution. Your patch seems to attempt a good thing, but it is tremendously big and complex (for me) to review. Can you separate a sequence of small patches? Note that "git log" is not what is wanted.

Some hints:

- Please try to minimize a patch.
 - It would be good to use the existing code as possible as you can.
 - It would be good not to change the existing code without reason.
- Please use 4 space for indent, with 8 space tab. (Emacs-style)
- Please use C89. Please don't use // comments.
- Please don't export function without "rb_" prefix, to avoid symbol conflicts.
- Why did you remove enumerator.so from "\$" and add lib/enumerator.rb? I'm afraid if it causes compatibility issue.
- It is slightly disturbing to define a new class with Array inherited. Is it really needed? Is there no alternative approach?

Thanks,

--

Yusuke Endoh mame@tsg.ne.jp

#20 - 05/30/2011 12:29 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 28-05-2011 18:38, Xavier Shay escreveu:

Issue [#3924](#) has been updated by Xavier Shay.

Rodrigo: Please try the latest version of rvm: rvm get head. Otherwise let me know what platform you are on.

Hi Xavier, I'm always on latest RVM. I don't know yet why ruby-head is resolving to 1.8.7-p334... But I don't intend to debug RVM for now. I'll try to compile Ruby manually when I have some time...

I'm on Debian unstable.

#21 - 05/30/2011 12:53 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 29-05-2011 12:23, Rodrigo Rosenfeld Rosas escreveu:

Em 28-05-2011 18:38, Xavier Shay escreveu:

Issue [#3924](#) has been updated by Xavier Shay.

Rodrigo: Please try the latest version of rvm: rvm get head. Otherwise let me know what platform you are on.

Hi Xavier, I'm always on latest RVM. I don't know yet why ruby-head is resolving to 1.8.7-p334... But I don't intend to debug RVM for now. I'll try to compile Ruby manually when I have some time...

I'm on Debian unstable.

Never mind. It seems the problem is that I tried installing 1.8.7 once but couldn't due to this issue:

<http://redmine.ruby-lang.org/issues/4556>

Then, ruby-head was always resolving to it. Now that I've applied the patch provided on issue manually to ~/.rvm/src, I could install 1.8.7 (needed for developing Gitorious). After installing it, ruby-head is resolving correctly. I'll try it and let you know.

#22 - 05/30/2011 07:42 AM - xaviershay (Xavier Shay)

Yusuke Endoh wrote:

Hello, Xavier

Thank you for your contribution. Your patch seems to attempt a good thing, but it is tremendously big and complex (for me) to review. Can you separate a sequence of small patches? Note that "git log" is not what is wanted.

I will do this.

Some hints:

- Please try to minimize a patch.
 - It would be good to use the existing code as possible as you can. I realise this would be ideal, but I can't figure out how it works, or whether it would be compatible with the new algorithm.
 - It would be good not to change the existing code without reason. Agree, but I am hoping the speed up is sufficient reason. To alleviate I have tried to make the code easier to read than the old version.
- Please use 4 space for indent, with 8 space tab. (Emacs-style)
- Please use C89. Please don't use // comments.
- Please don't export function without "rb_" prefix, to avoid symbol conflicts.
I will do all of these.
- Why did you remove enumerator.so from "\$" and add lib/enumerator.rb?
I'm afraid if it causes compatibility issue.
provide doesn't really make sense and is confusing. It's saying "Pretend this file has been loaded", and then we need to make allowances for files that don't really exist, which means lots of extra checking. Providing lib/enumerator.rb as a blank file obviates this problem.
- It is slightly disturbing to define a new class with Array inherited.
Is it really needed? Is there no alternative approach?
I agree and wish there was a good alternative. Unfortunately this is necessary because \$LOADED_FEATURES is treated like an array by much existing code (particularly tests), but we need it as a case-insensitive hash for the new algorithm. Perhaps the hash could be exposed directly as a class (like CaseInsensitiveHash), but this means more code, is potentially more complicated, and means we are really changing the type of \$LOADED_FEATURES. Thoughts?

Thanks,

--

Yusuke Endoh mame@tsg.ne.jp

#23 - 05/30/2011 04:33 PM - shyouhei (Shyouhei Urabe)

=begin

Hi Xavier. I'm also reading your patch.

- Though the patch is big, after applying it's super readable than the original. Fairly good.
 - I also like your writing static function declarations.
- Style things are already pointed out by Yusuke so I don't repeat.
- As for the enumerator thing, I think it's worth considering your idea. That enumerator hack is for backward compatibility, so in a future when we don't need that hack anymore, we can just delete that lib/enumerator.rb. I think it's cool.
- I'm not sure if your code is ready for case-insensitive filesystems like those in Windows. I'm sure you know the problem so I should have missed something.

=end

#24 - 05/30/2011 10:59 PM - mame (Yusuke Endoh)

Hello,

2011/5/30 Xavier Shay xavier-list@rhn.net:

- Please try to minimize a patch. Å - It would be good to use the existing code as possible as you can. I realise this would be ideal, but I can't figure out how it works, or whether it would be compatible with the new algorithm.

Indeed, load.c is very complex, but I believe that this is because load.c is an accumulation of bad know-hows in the long history. So we should not throw it away without understanding, I think.

- It is slightly disturbing to define a new class with Array inherited. \hat{A} Is it really needed? \hat{A} Is there no alternative approach? I agree and wish there was a good alternative. Unfortunately this is necessary because \$LOADED_FEATURES is treated like an array by much existing code (particularly tests), but we need it as a case-insensitive hash for the new algorithm. Perhaps the hash could be exposed directly as a class (like CaseInsensitiveHash), but this means more code, is potentially more complicated, and means we are really changing the type of \$LOADED_FEATURES. Thoughts?

Aha, I understand why the inheritance is used. It is to hook the array modification, right?

It is an impossible approach because some extension library can modify \$LOADED_FEATURES directly by rb_ary_push. I found amalgalite to do so actually:

<http://www.google.com/codesearch/p?hl>

#25 - 05/31/2011 12:23 AM - mame (Yusuke Endoh)

Hello, Xavier

<http://rnh.net/2011/05/28/speeding-up-rails-startup-time>

In the above article, you said that the bottleneck is to find \$LOADED_FEATURES in linear, but I doubt it. If you are right, 1.8 should be also slow because 1.8 uses the same algorithm. Thus, I guess there is another bottleneck.

I tried your patch and test in <https://gist.github.com/985224> on Ubuntu.

I could confirm that trunk is twice slower than 1.9.2p180.

But I couldn't confirm that the speed up by your patch.

I also performed the same benchmark after "gem install rails", but there is no difference.

1.9.2p180

\$ ruby -v full_load_path_benchmark.rb

ruby 1.9.2p180 (2011-02-18 revision 30909) [i686-linux]

user system total real

0 in load path 0.000000 0.000000 0.000000 (0.003764)

500 in load path 0.040000 0.040000 0.080000 (0.081142)

1000 in load path 0.060000 0.100000 0.160000 (0.159599)

1500 in load path 0.130000 0.100000 0.230000 (0.235893)

2000 in load path 0.120000 0.200000 0.320000 (0.317750)

trunk

\$../ruby.org -v full_load_path_benchmark.rb

ruby 1.9.3dev (2011-05-30 trunk 31824) [i686-linux]

user system total real

0 in load path 0.000000 0.000000 0.000000 (0.005887)

500 in load path 0.090000 0.110000 0.200000 (0.208288)

1000 in load path 0.160000 0.250000 0.410000 (0.406125)

1500 in load path 0.200000 0.370000 0.570000 (0.586779)

2000 in load path 0.240000 0.500000 0.740000 (0.749551)

Xavier patch

\$../ruby.new -v full_load_path_benchmark.rb

ruby 1.9.3dev (2011-05-30 trunk 31824) [i686-linux]

user system total real

0 in load path 0.000000 0.000000 0.000000 (0.006294)

500 in load path 0.080000 0.210000 0.290000 (0.293921)

1000 in load path 0.190000 0.390000 0.580000 (0.585095)

1500 in load path 0.290000 0.540000 0.830000 (0.839902)

2000 in load path 0.370000 0.740000 1.110000 (1.114302)

--

Yusuke Endoh mame@tsg.ne.jp

#26 - 05/31/2011 01:59 AM - mame (Yusuke Endoh)

Hello,

2011/5/31 Yusuke ENDOH mame@tsg.ne.jp:

Hello, Xavier

<http://rnh.net/2011/05/28/speeding-up-rails-startup-time>

In the above article, you said that the bottleneck is to find \$LOADED_FEATURES in linear, but I doubt it. If you are right, 1.8 should be also slow because 1.8 uses the same algorithm. Thus, I guess there is another bottleneck.

I tried your patch and test in <https://gist.github.com/985224> on Ubuntu.

I could confirm that trunk is twice slower than 1.9.2p180.

I investigated this issue with ko1 and tarui-san.

Trunk seems to create more objects during require than 1.9.2. These objects are not collected because the benchmark program stops it by GC.disable.

1.9.2: 0.30 sec (w/ GC.disable) -> 0.32 sec (w/o GC.disable)
trunk: 0.73 sec (w/ GC.disable) -> 0.48 sec (w/o GC.disable)

The first issue is in the benchmark program.

Still, trunk is 1.5x slower than 1.9.2 because of the object generation.

Each require does the something like this:

```
$LOADED_FEATURES.map {|f| File.expand_path(f) }
```

This process creates many objects, i.e., strings. Typically, \$LOADED_FEATURES are already expanded, so the process is not needed in normal cases. In fact, 1.9.2 expands the paths only when they are not absolute, like this:

```
$LOADED_FEATURES.map {|f| File.expand_path(f) if f is not absolute }
```

However, the check was removed at r30789, I cannot understand the reason of the change because the commit log is too quiet, but I found [ruby-core:34593] and guess that it motivated the change:

Autoload treatment of absolute paths in \$LOAD_PATH containing . or ..

I think reverting r30789 is an good solution in the immediate future.

--
Yusuke Endoh mame@tsg.ne.jp

#27 - 05/31/2011 02:29 AM - mame (Yusuke Endoh)

Hello,

I'd like to show a new benchmark on Ubuntu.

- trunk becomes as fast as 1.9.2p180 if r30789 is reverted
- 1.8.7 is slower than 1.9.2p180 on my environment

Sorry for my sending many mails in a short time.

```
# 1.9.2p180
$ ruby -v full_load_path_benchmark.rb
ruby 1.9.2p180 (2011-02-18 revision 30909) [i686-linux]
user system total real
0 in load path 0.010000 0.000000 0.010000 ( 0.004401)
```

```
500 in load path 0.030000 0.040000 0.070000 ( 0.097179)
1000 in load path 0.070000 0.110000 0.180000 ( 0.190188)
1500 in load path 0.130000 0.120000 0.250000 ( 0.271191)
2000 in load path 0.110000 0.230000 0.340000 ( 0.377725)
```

trunk with r30789 reverted

```
$. /ruby -v full_load_path_benchmark.rb
ruby 1.9.3dev (2011-05-30 trunk 31824) [i686-linux]
user system total real
0 in load path 0.010000 0.000000 0.010000 ( 0.010032)
500 in load path 0.020000 0.050000 0.070000 ( 0.101144)
1000 in load path 0.050000 0.120000 0.170000 ( 0.189199)
1500 in load path 0.100000 0.150000 0.250000 ( 0.275388)
2000 in load path 0.130000 0.200000 0.330000 ( 0.363190)
```

1.8.7p302 (Ubuntu package)

```
$/usr/bin/ruby -v full_load_path_benchmark.rb
ruby 1.8.7 (2010-08-16 patchlevel 302) [i686-linux]
user system total real
0 in load path 0.000000 0.000000 0.000000 ( 0.005125)
500 in load path 0.030000 0.090000 0.120000 ( 0.142067)
1000 in load path 0.080000 0.170000 0.250000 ( 0.271456)
1500 in load path 0.090000 0.290000 0.380000 ( 0.419044)
2000 in load path 0.140000 0.400000 0.540000 ( 0.560379)
```

1.8.7p334 (self-built with -O2)

```
$/ruby-1.8.7-p334 -v full_load_path_benchmark.rb
ruby 1.8.7 (2011-02-18 patchlevel 334) [i686-linux]
user system total real
0 in load path 0.000000 0.000000 0.000000 ( 0.002295)
500 in load path 0.040000 0.080000 0.120000 ( 0.114809)
1000 in load path 0.060000 0.160000 0.220000 ( 0.226771)
1500 in load path 0.130000 0.210000 0.340000 ( 0.337815)
2000 in load path 0.130000 0.330000 0.460000 ( 0.455198)
```

--

Yusuke Endoh mame@tsg.ne.jp

#28 - 05/31/2011 07:23 AM - xaviershay (Xavier Shay)

On 30/05/11 11:55 PM, Yusuke ENDOH wrote:

Hello,

2011/5/30 Xavier Shay xavier-list@rnh.net:

- Please try to minimize a patch.
 - It would be good to use the existing code as possible as you can. I realise this would be ideal, but I can't figure out how it works, or whether it would be compatible with the new algorithm.

Indeed, load.c is very complex, but I believe that this is because load.c is an accumulation of bad know-hows in the long history. So we should not throw it away without understanding, I think. This is a fair point. Perhaps a good first step would be trying to refactor it with better variable and method names so that the intent of the code can reveal itself.

- It is slightly disturbing to define a new class with Array inherited. Is it really needed? Is there no alternative approach? I agree and wish there was a good alternative. Unfortunately this is necessary because \$LOADED_FEATURES is treated like an array by much existing code (particularly tests), but we need it as a case-insensitive hash for the new algorithm. Perhaps the hash could be exposed directly as a class (like CaseInsensitiveHash), but this means more code, is potentially more complicated, and means we are really changing the type of \$LOADED_FEATURES. Thoughts?

Aha, I understand why the inheritance is used. It is to hook the array modification, right?
correct

It is an impossible approach because some extension library can modify \$LOADED_FEATURES directly by rb_ary_push.
I found amalgalite to do so actually:

http://www.google.com/codesearch/p?hl=ja#MyfZfO3_1cw/ext/amalgalite/amalgalite3_requires_bootstrap.c&q=LOADED_FEATURES%20lang:c%20rb_ary_push&l=151

```
rb_ary_push( rb_gv_get( "$LOADED_FEATURES" ), require_name );
```

It seems that amalgalite is the only extension to do this in that search. While I realise that doesn't cover everything, it seems the impact is perhaps small? Would it be possible to instead work with the maintainer to fix, and publicize this change. It seems an odd API to be supporting.

Xavier

#29 - 05/31/2011 07:23 AM - xaviershay (Xavier Shay)

On 30/05/11 5:35 PM, Shyouhei Urabe wrote:

- I'm not sure if your code is ready for case-insensitive filesystems like those in Windows. I'm sure you know the problem so I should have missed something. I don't have a Windows environment to test with. I would be most appreciative if anyone was able to try it out.

Xavier

#30 - 05/31/2011 07:23 AM - xaviershay (Xavier Shay)

On 31/05/11 1:16 AM, Yusuke ENDOH wrote:

Hello, Xavier

<http://rhn.net/2011/05/28/speeding-up-rails-startup-time>

In the above article, you said that the bottleneck is to find \$LOADED_FEATURES in linear, but I doubt it. If you are right, 1.8 should be also slow because 1.8 uses the same algorithm. Thus, I guess there is another bottleneck. 1.8 actually does exhibit the same performance curve, just over a greater value of N. See this graph (x-axis doubled from other benchmarks)

<https://gist.github.com/c8d0d422a9203e1fe492#gistcomment-30971>

I don't doubt there are other bottlenecks too though.

I tried your patch and test in <https://gist.github.com/985224> on Ubuntu. I could confirm that trunk is twice slower than 1.9.2p180. But I couldn't confirm that the speed up by your patch. I also performed the same benchmark after "gem install rails", but there is no difference. In that case perhaps it is related to the CASEFOLD_FILESYSTEM? I have been benchmarking on OSX.

A related change was reverted recently in r31692.

Cheers,
Xavier

#31 - 05/31/2011 07:23 AM - luislavena (Luis Lavena)

On Mon, May 30, 2011 at 6:00 PM, Xavier Shay xavier-list@rhn.net wrote:

It is an impossible approach because some extension library can modify \$LOADED_FEATURES directly by rb_ary_push. I found amalgalite to do so actually:

<http://www.google.com/codesearch/p?hl>

#32 - 05/31/2011 07:23 AM - luislavena (Luis Lavena)

On Mon, May 30, 2011 at 6:01 PM, Xavier Shay xavier-list@rhn.net wrote:

On 30/05/11 5:35 PM, Shyouhei Urabe wrote:

- I'm not sure if your code is ready for case-insensitive filesystems like those in Windows. I'm sure you know the problem so I should have missed something.

I don't have a Windows environment to test with. I would be most appreciative if anyone was able to try it out.

I do have a Windows system and I've provided my initial numbers.

However, as was previously stated by other core members, the size of the patch is hard to process and analyze properly, which might not catch corner cases that were present in load.c and this newer code do not contemplate.

For me to completely test this on Windows will take more than a week, as I have multiple applications with different requirements and to properly perform this I need to track pre and post performance details of each of those applications to provide fair results.

--

Luis Lavena
AREA 17

-

Perfection in design is achieved not when there is nothing more to add, but rather when there is nothing more to take away.
Antoine de Saint-Exupéry

#33 - 05/31/2011 09:29 AM - mame (Yusuke Endoh)

Hello,

2011/5/31 Xavier Shay xavier-list@rhn.net:

```
rb_ary_push( rb_gv_get( "$LOADED_FEATURES" ), require_name );
```

It seems that amalgalite is the only extension to do this in that search. While I realise that doesn't cover everything, it seems the impact is perhaps small? Would it be possible to instead work with the maintainer to fix, and publicize this change. It seems an odd API to be supporting.

I don't think so. When I try to add a path into \$LOADED_FEATURES by extension library, rb_ary_push is the first way that came to mind.

2011/5/31 Xavier Shay xavier-list@rhn.net:

1.8 actually does exhibit the same performance curve, just over a greater value of N. See this graph (x-axis doubled from other benchmarks)

Hmm... Okay. However, it is hard to fix this issue in 1.9.x, I think.

To be honest, \$LOADED_FEATURES as an array is uncool, or even considered as a "specification bug", though.

I tried your patch and test in <https://gist.github.com/985224> on Ubuntu.
I could confirm that trunk is twice slower than 1.9.2p180.
But I couldn't confirm that the speed up by your patch.
I also performed the same benchmark after "gem install rails", but there is no difference.

In that case perhaps it is related to the CASEFOLD_FILESYSTEM? I have been benchmarking on OSX.

I see!

A related change was reverted recently in r31692.

Then, is there no problem in os x now?

--

Yusuke Endoh mame@tsg.ne.jp

#34 - 05/31/2011 09:08 PM - xaviershay (Xavier Shay)

Hello,

This is a long message, but I have tried to address most of the concerns with my patch. There are two sections: one for the technical detail of the patch, and one for benchmarks.

= The patch

I have split my patch up into a six smaller patches which I hope are easier to review.

Descriptions: <https://gist.github.com/35060fbcefb25cf1a456>

Patches: <https://gist.github.com/58dbd6e72c1a1f47a415>

GitHub: <https://github.com/xaviershay/ruby/commits/require-patch>

In addition I have addressed the following concerns from Yusuke:

- Please use 4 space for indent, with 8 space tab. (Emacs-style)
- Please use C89. Please don't use // comments.
- Please don't export function without "rb_" prefix, to avoid symbol conflicts.

I think there are still two big concerns:

- 1) Significantly different from original load.c
- 2) Compatibility with Windows
- 3) LoadedFeaturesProxy

Regarding #1, I don't have anything further to add at the moment.

Regarding #2, it will need to be tested thoroughly like all other platforms, but I don't believe it would result in significant architectural changes.

For #3, I was worried about it but on reflection I think it may not be a problem.

Yusuke wrote:

"It is an impossible approach because some extension library can modify \$LOADED_FEATURES directly by rb_ary_push."

The hooks are only used to keep a cache up to date. If we get a cache-miss, we could just revert to scanning \$LOADED_FEATURES again which would pick up any items that were added with rb_ary_push.

This will degrade worst-case performance, but I don't think that matters because most requires will succeed.

Does this make sense?

This would also mean the addition of lib/enumerator.rb in my patch would be unnecessary (though IMO probably still a good idea).

= Benchmarks

Yusuke I think I mislead you with the benchmarks. I do not expect full_load_path_benchmark.rb to be faster. I was only using it to ensure I didn't make that case any worse. See below for further details.

Here are my updated timings, with descriptions of each. For the load path and require benchmark I have only included the final number in the test to allow for a more focused comparison.

Run on OSX 10.7

	1.9.2p180	1.9.3*	1.9.3**	1.9.3***
load path	0.667376	0.866228	0.623411	0.699315
requires	6.745875	7.921598	7.261326	0.285119
new rails	2.172	1.412	1.624	1.082
medium rails	18.372	17.59	15.855	10.488

- 1.9.3r31827 (31/5) ** 1.9.3r31827, with r30789 reverted *** 1.9.3r31827, with my patch

== Benchmark descriptions

load path - <https://gist.github.com/985224>

Loading a file 50 times with 2000 entries in the \$LOAD_PATH.

This is an academic benchmark, since 2000 entries is a large amount. For comparison, our rails app load path only has 42 entries.

Performance for all versions is linear and roughly equivalent.

Note that I removed the GC.disable that was in earlier versions of this benchmark.

requires - <https://gist.github.com/c8d0d422a9203e1fe492>

Loading 2500 files. This is a more relevant benchmark. Our Rails app loads ~2200 files, large apps can load as many as 9000 [1].

All versions display an exponential increase in time as N increases. 1.8.7, though not included in this measurement, also displays such an exponential curve but it isn't as noticeable in comparison with 1.9 since the magnifier is far less [2].

[1] sent to me via private correspondence

[2] see <https://gist.github.com/c8d0d422a9203e1fe492>

new rails

Using rails 3.0.7:

```
rails new test-rails-app
cd test-rails-app
time ruby script/rails runner "puts 1"
```

medium rails

This is my main Rails code base. Sorry I cannot share it, though this benchmark is not required to make a case for this patch since it shows roughly the same percentage improvement as a blank rails app.

Thanks,
Xavier

#35 - 05/31/2011 09:13 PM - xaviershay (Xavier Shay)

Argh I mangled my benchmark table. Here is a picture that is easy to read:

<https://img.skitch.com/20110531-qn7nukkyretiepysmytm8r1757.jpg>

#36 - 05/31/2011 11:24 PM - mame (Yusuke Endoh)

Hello,

2011/5/31 Xavier Shay xavier-list@rhn.net:

Patches: <https://gist.github.com/58dbd6e72c1a1f47a415>

Awesome!

In addition I have addressed the following concerns from Yusuke:

- Please use 4 space for indent, with 8 space tab. (Emacs-style)

Your patch still seems to use 8-space (1-tab) indent.

Yours:

```
+static VALUE
+rb_locate_file(VALUE filename)
+{
  • VALUE full_path = Qnil; ~~~~~~ 1 tab
```

Expected:

```
+static VALUE
+rb_locate_file(VALUE filename)
+{
  • VALUE full_path = Qnil; ~~~~ 4 spaces
```

But okay, this is a minor problem. We can fix it easily later.

Yusuke wrote:

"It is an impossible approach because some extension library can modify \$LOADED_FEATURES directly by rb_ary_push."

The hooks are only used to keep a cache up to date. If we get a cache-miss, we could just revert to scanning \$LOADED_FEATURES again which would pick up any items that were added with rb_ary_push.

Sorry I cannot get your point.

The following program should not load t.rb, should it?

```
[$$.size] = File.expand_path("./t.rb")
require("./t")
```

However, this actually load t.rb after your patch is applied:

```
$ cat t.rb
p :loaded
```

```
$.ruby -I. -e '
[$$.size] = File.expand_path("./t.rb")
require("./t")
'
:loaded
```

This is considerable incompatibility, I think.

Of course, you can change LoadedFeaturesProxy to hook Array#[]=.
But you cannot hook rb_ary_push. This is my concern.

This would also mean the addition of lib/enumerator.rb in my patch would be unnecessary (though IMO probably still a good idea).

Yes, it is a good idea, and will be accepted eventually.
But I'm afraid if it is not the timing to do so.
And we should discuss it separately.

= Benchmarks

Yusuke I think I mislead you with the benchmarks. I do not expect full_load_path_benchmark.rb to be faster. I was only using it to ensure I didn't make that case any worse. See below for further details.

Okay, I got it. Sorry for my obtuseness.
However, I don't think that the incompatibility is ignorable.
At least, I have no right to decide it.

Matz and Yugui, what do you think?

--

Yusuke Endoh mame@tsg.ne.jp

#37 - 06/01/2011 07:17 AM - tarui (Masaya Tarui)

Hello,

Require performance has been improved a little at r31875, i think.
Please compare the proposal with it.

Thank you.

Yusuke wrote:

2011/5/31 Xavier Shay :

1.8 actually does exhibit the same performance curve, just over a greater value of N. See this graph (x-axis doubled from other benchmarks)
Hmm... Okay. However, it is hard to fix this issue in 1.9.x,
I think.
To be honestly, \$LOADED_FEATURES as an array is uncool, or even considered as a "specification bug", though.

I agree with you.

#38 - 06/01/2011 07:59 AM - xaviershay (Xavier Shay)

On 1/06/11 12:24 AM, Yusuke Endoh wrote:

In addition I have addressed the following concerns from Yusuke:

- Please use 4 space for indent, with 8 space tab. (Emacs-style)

Your patch still seems to use 8-space (1-tab) indent.

Sorry I misunderstood what you were asking for. Too long working with ruby 2 spaces, I have forgotten what tabbing looks like :)

After other things are resolved I will go through and ensure all of load.c is formatted correctly.

Yusuke wrote:

"It is an impossible approach because some extension library can modify \$LOADED_FEATURES directly by rb_ary_push."

The hooks are only used to keep a cache up to date. If we get a cache-miss, we could just revert to scanning \$LOADED_FEATURES again which would pick up any items that were added with rb_ary_push.

Sorry I cannot get your point.

The following program should not load t.rb, should it?

```
[$$.size] = File.expand_path("./t.rb")
require("./t")
```

However, this actually load t.rb after your patch is applied:

```
$ cat t.rb
p :loaded
```

```
$.ruby -I. -e '
[$$.size] = File.expand_path("./t.rb")
require("./t")
'
:loaded
```

This is considerable incompatibility, I think.

Of course, you can change LoadedFeaturesProxy to hook Array#[]=.

But you cannot hook rb_ary_push. This is my concern.

I was thinking of the case where non-file features are pushed to the array (such as enumerator.so and almagamite).

```
rb_ary_push($LOADED_FEATURES, 'enumerator.so')
require 'enumerator'
```

Perhaps a scan of the load path to cover this case wouldn't be too bad? (Not in my patch, just thinking out loud).

This does not cover your case though so perhaps not a great idea:

```
rb_ary_push($LOADED_FEATURES, File.expand_path("./t"))
require "./t"
```

Two other options that maybe you will like:

1) store some metadata against entries in \$LOADED_FEATURES to indicate whether they have been cached. rb_ary_push will not set this, so a quick scan can be made of \$LOADED_FEATURES on require to see if a recache is needed

2) On require, recache if the count of \$LOADED_FEATURES does not match the count of the cache. This isn't 100% correct --- you could rb_ary_pop then rb_ary_push to work around it --- but perhaps if more acceptable?

I think #2 if the limitation is acceptable. Items added by rb_ary_push would be located on the filesystem if possible (such as ".t" in your example), other wise left as is (such as "enumerator.so").

This would also mean the addition of lib/enumerator.rb in my patch would be unnecessary (though IMO probably still a good idea).

Yes, it is a good idea, and will be accepted eventually.

But I'm afraid if it is not the timing to do so.
And we should discuss it separately.
ok

Thanks,
Xavier

#39 - 06/01/2011 09:23 AM - shyouhei (Shyouhei Urabe)

Hi Yusuke,

(05/31/2011 01:58 AM), Yusuke ENDOH wrote:

Still, trunk is 1.5x slower than 1.9.2 because of the object generation.
Each require does the something like this:

```
$LOADED_FEATURES.map {|f| File.expand_path(f) }
```

This process creates many objects, i.e., strings. Typically, \$LOADED_FEATURES are already expanded, so the process is not needed in normal cases. In fact, 1.9.2 expands the paths only when they are not absolute, like this:

```
$LOADED_FEATURES.map {|f| File.expand_path(f) if f is not absolute }
```

If it's the actual problem, why not cache the expanded path?

<https://github.com/shyouhei/ruby/commit/c229cb4>

#40 - 06/03/2011 04:53 AM - xaviershay (Xavier Shay)

On 31/05/11 3:55 PM, Xavier Shay wrote:

Two other options that maybe you will like:

- 1) store some metadata against entries in \$LOADED_FEATURES to indicate whether they have been cached. rb_ary_push will not set this, so a quick scan can be made of \$LOADED_FEATURES on require to see if a recache is needed
- 2) On require, recache if the count of \$LOADED_FEATURES does not match the count of the cache. This isn't 100% correct --- you could rb_ary_pop then rb_ary push to work around it --- but perhaps if more acceptable?

I think #2 if the limitation is acceptable. Items added by rb_ary_push would be located on the filesystem if possible (such as ".t" in your example), other wise left as is (such as "enumerator.so").

I have put together a proof of concept of this:

<https://gist.github.com/8bb23db32d861cbe952f>

It removes the need for both lib/enumerator.rb, and the changes I made to loading mkmf in ext/exmke.rb

I still need to test it with amalgamite (was on a plane so didn't have internet to get it).

If you think the limitations are ok, I will work it back into my other patch set.

Also, I note a change to load.c has just gone into trunk related to performance. I will add this to my list of benchmarks in the next few days.

Cheers,
Xavier

#41 - 06/05/2011 03:23 PM - xaviershay (Xavier Shay)

On 1/06/11 8:17 AM, Masaya Tarui wrote:

Issue [#3924](#) has been updated by Masaya Tarui.

Hello,

Require performance has been improved a little at r31875, i think.
nice! It sure has.

Please compare the proposal with it

```
Mine 1.9.3r31923
```

```
2000 in load path 0.70 0.702834
2500 requires 0.29 5.676016
new rails app 1.08 1.346
medium rails app 10.49 10.88
```

The underlying algorithm is still problematic (see 2500 requires benchmark), but I don't have strong evidence that it is affecting performance of real apps given it appears to perform well on rails applications. Assuming no contrary evidence appears, and without understanding this difference further, you probably won't want to apply my patch for a 1.9.3 point release given the risk of regressions. I still think it should be given consideration for a future major release though.

Out of interest ... how does your commit work? Is it an optimization perhaps my patch could benefit from? My concern with r31875 is that it makes that loop even harder to understand for those not as familiar with that part of the code.

Cheers,
Xavier

Thank you.

Yusuke wrote:

2011/5/31 Xavier Shay :

1.8 actually does exhibit the same performance curve, just over a greater value of N. See this graph (x-axis doubled from other benchmarks)
Hmm... Okay. However, it is hard to fix this issue in 1.9.x,
I think.
To be honestly, \$LOADED_FEATURES as an array is uncool, or even considered as a "specification bug", though.

I agree with you.

Bug #3924: Performance bug (in require?)
<http://redmine.ruby-lang.org/issues/3924>

Author: Carsten Bormann
Status: Open
Priority: Normal
Assignee:
Category: core
Target version: 1.9.3
ruby -v: -

```
=begin
Running irb< /dev/null in 1.9.2 causes 3016 calls to lstat64.
```

For instance, there is a sequence of 28 repetitions each of lstat calls to all 6 non-empty path prefixes of /opt/local/lib/ruby1.9/1.9.1/irb.rb -- a total of 170 lstats apparently just to load this file; another set of lstats then occurs later for another 18 (times 6) times. Clearly, something is running amok in the calling sequence rb_require_safe -> realpath_rec -> lstat.

Another example: Running a simple test with the baretest gem causes 17008 calls to lstat. According to perftools.rb, 80 % of the 1.2 seconds of CPU is used in Kernel#gem_original_require (and another 12 in GC, some of which may be caused by this).

```
=end
```

#42 - 06/09/2011 05:23 AM - tarui (Masaya Tarui)

Require performance has been improved a little at r31875, i think.

nice! It sure has.

Please compare the proposal with it

Â Â Â Â Â Â Â Â Â Â Mine Â Â 1.9.3r31923
2000 in load path Â Â Â 0.70 Â Â 0.702834
2500 requires Â Â Â Â Â 0.29 Â Â 5.676016
new rails app Â Â Â Â Â 1.08 Â Â 1.346
medium rails app Â Â Â Â Â 10.49 Â 10.88

The underlying algorithm is still problematic (see 2500 requires benchmark), but I don't have strong evidence that it is affecting performance of real apps given it appears to perform well on rails applications. Assuming no contrary evidence appears, and without understanding this difference further, you probably won't want to apply my patch for a 1.9.3 point release given the risk of regressions. I still think it should be given consideration for a future major release though.

Thank you for benchmarking. (I don't have rails benchmarks.)
It is good news that it's effective enough for rails.

I think that your idea is good, and heard Hash is already used in JRuby.
But your patch is large and has some compatibility issue,
so I'm afraid for reject at 1.9.3.

Out of interest ... how does your commit work? Is it an optimization perhaps my patch could benefit from? My concern with r31875 is that it makes that loop even harder to understand for those not as familiar with that part of the code.

In loaded_feature_path and outer of it,
it takes the match LOADED_FEATURE and LOAD_PATH+target in each of LOADED_FEATURES and LOAD_PATH array for back compatibility (i think) .
before patch, part of matching LOADED_FEATURE and target is in inner loop,
so I only let it out.
LOADED_FEATURE is not matched with target with many case, and patch makes return fast.

In your patch, cache processing is similar to its feature, and already optimized.

Regards,
Masaya

#43 - 06/11/2011 04:43 PM - ko1 (Koichi Sasada)

- Status changed from Open to Assigned
- Assignee set to yugui (Yuki Sonoda)

#44 - 06/20/2011 06:01 PM - jskrzypek (Jarosław Skrzypek)

You might be also interested in my very short patch: <https://gist.github.com/1035322> Basically it reorders conditions to start with fastest ones and execute slower ones only if needed. Here is also some description:
<http://www.lunarlogicpolska.com/blog/2011/06/14/tracing-processes-for-fun-and-profit.html>

#45 - 06/23/2011 12:23 PM - nahi (Hiroshi Nakamura)

Hi all,

On Wed, Jun 1, 2011 at 09:14, Urabe Shyouhei shyouhei@ruby-lang.org wrote:

This process creates many objects, i.e., strings. Â Typically, \$LOADED_FEATURES are already expanded, so the process is not needed in normal cases. Â In fact, 1.9.2 expands the paths only when they are not absolute, like this:

```
Â $LOADED_FEATURES.map {|f| File.expand_path(f) if f is not absolute }
```

If it's the actual problem, why not cache the expanded path?

<https://github.com/shyouhei/ruby/commit/c229cb4>

I did a benchmark > <https://gist.github.com/1041791> (full)

Measured loading time of 2 Rails 3.0.7 apps

- emptyApp: script/rails generate emptyApp
- slow-rails: by Joe Van Dyk (<https://github.com/joelvandyk/slow-rails>)

Interpreters

- ruby 1.9.2p274 (2011-06-06 revision 31932) [x86_64-linux]
- ruby 1.9.3dev (2011-06-22 trunk 32204) [x86_64-linux]
- ruby 1.9.3dev (2011-06-22 trunk 32204) [x86_64-linux] + Shyouhei's expand_path cache patch (<https://github.com/shyouhei/ruby/commit/c229cb4>)

Results (average wall clock time of 'time ruby script/rails runner 0' 10 times)

- 1.9.2p274
 - emptyApp: 1.87 [sec]
 - slow-rails: 8.69 [sec]
- 1.9.3dev of today
 - emptyApp: 1.35 [sec](#)
 - slow-rails: 6.37 [sec](#)
- 1.9.3dev + Shyouhei's expand_path cache patch
 - emptyApp: 1.07 [sec](#)
 - slow-rails: 3.81 [sec](#)

Awesome result.

Anyone can imagine a downside of this? It could not work as expected if the result of `rb_file_expand_path` *changes* during require (adding a new file during require should work.) Do we need to care such a case?

Regards,
// NaHi

#46 - 06/23/2011 12:53 PM - mame (Yusuke Endoh)

Hi,

2011/6/23 Hiroshi Nakamura nakahiro@gmail.com:

Anyone can imagine a downside of this? It could not work as expected if the result of `rb_file_expand_path` *changes* during require (adding a new file during require should work.) Do we need to care such a case?

As far as I look the patch, paths that are once stored in cache will never be collected. It may be considered as a kind of memory leaks.

I guess that it is not a problem in casual cases. But I'm not sure if it is really OK.

--
Yusuke Endoh mame@tsg.ne.jp

#47 - 06/23/2011 07:53 PM - nahi (Hiroshi Nakamura)

Hi,

2011/6/20 Jarosław Skrzypek skrzypek.jarek@gmail.com:

You might be also interested in my very short patch: <https://gist.github.com/1035322> Basically it reorders conditions to start with fastest ones and execute slower ones only if needed. Here is also some description: <http://www.lunarlogicpolska.com/blog/2011/06/14/tracing-processes-for-fun-and-profit.html>

Good perf improvement. Yugui-san, it could be a candidate for 1.9.2 medication, if the backporting 1.9.3dev patches are big to apply.

Regards,
// NaHi

#48 - 06/28/2011 06:17 AM - nahi (Hiroshi Nakamura)

- Status changed from Assigned to Closed

I close this ticket as 'Duplicated'. Please refer [#3924](#).

#49 - 06/28/2011 04:20 PM - cabo (Carsten Bormann)

Well, you can't close both tickets as duplicates of each other. Which one stays open? The Windows one or this one, which has the discussion?

#50 - 06/28/2011 08:33 PM - nahi (Hiroshi Nakamura)

- Status changed from Closed to Open

Oops, I'm very sorry. I wanted to close only [#3906](#). Reopened it.

#51 - 07/11/2011 09:00 AM - kosaki (Motohiro KOSAKI)

- Status changed from Open to Assigned

Nahi-san, Yugui-san, May I ask current status of this?

#52 - 07/11/2011 10:37 PM - nahi (Hiroshi Nakamura)

Issues left are;

- Find a way to merge Xavier's work.
- Merge Shyouhei's optimization by cache.

I suggest to change Target version to 1.9.X.

#53 - 07/11/2011 10:58 PM - antares (Michael Klishin)

I second Hiroshi's suggestion to solve this for 1.9.x and not just 1.9.3.

#54 - 07/11/2011 11:05 PM - svenfuchs (Sven Fuchs)

+1 on 1.9x

#55 - 07/13/2011 06:53 PM - Eregon (Benoit Daloze)

On 11 July 2011 15:58, Michael Klishin redmine@ruby-lang.org wrote:

Issue [#3924](#) has been updated by Michael Klishin.

I second Hiroshi's suggestion to solve this for 1.9.x and not just 1.9.3.

1.9.x means it should be solved for next 1.9, so 1.9.4 or later, it is not about backporting.

It can not be merged in 1.9.3, because it is late.

#56 - 07/21/2011 02:02 PM - kosaki (Motohiro KOSAKI)

- Target version changed from 1.9.3 to 2.0.0

I've switched the target version to 1.9.x. This issue is absolutely important, but sadly it's too late. ;-)

#57 - 02/12/2012 12:20 PM - shiba (satoshi shiba)

Hi,

Yusuke ENDOH wrote:

Hi,

2011/6/23 Hiroshi Nakamura nakahiro@gmail.com:

Anyone can imagine a downside of this? It could not work as expected if the result of `rb_file_expand_path` *changes* during require (adding a new file during require should work.) Do we need to care such a case?

As far as I look the patch, paths that are once stored in cache will never be collected. It may be considered as a kind of memory leaks.

I guess that it is not a problem in casual cases. But I'm not sure if it is really OK.

May I ask current status of this problem?

"`rb_file_expand_path()`" is still a bottleneck of "`rb_require_safe()`".

So, I'm interested in this problem.

If memory leak is a problem, how about collecting strings at the time of mark phase.

I made a patch which is based on Shyouhei's patch to resolve memory leak.

A patch(<https://gist.github.com/1805919>) collects cached strings at the time of mark phase.

#58 - 02/13/2012 09:29 AM - luislavena (Luis Lavena)

satoshi shiba wrote:

May I ask current status of this problem?

"`rb_file_expand_path()`" is still a bottleneck of "`rb_require_safe()`".

So, I'm interested in this problem.

If memory leak is a problem, how about collecting strings at the time of mark phase.

I made a patch which is based on Shyouhei's patch to resolve memory leak.

A patch(<https://gist.github.com/1805919>) collects cached strings at the time of mark phase.

Please see Feature [#5767](#), proposed usage of cached expanded paths:

<https://bugs.ruby-lang.org/issues/5767>

#59 - 02/13/2012 09:43 AM - shiba (satoshi shiba)

Luis Lavena wrote:

satoshi shiba wrote:

May I ask current status of this problem?

"`rb_file_expand_path()`" is still a bottleneck of "`rb_require_safe()`".

So, I'm interested in this problem.

If memory leak is a problem, how about collecting strings at the time of mark phase.

I made a patch which is based on Shyouhei's patch to resolve memory leak.

A patch(<https://gist.github.com/1805919>) collects cached strings at the time of mark phase.

Please see Feature [#5767](#), proposed usage of cached expanded paths:

<https://bugs.ruby-lang.org/issues/5767>

Thank you for telling me.

I have missed that ticket.

#60 - 11/20/2012 09:40 AM - h.shirosaki (Hiroshi Shirosaki)

- Status changed from Assigned to Closed

Fixed by [#7158](#).

Files

dt1	285 KB	10/10/2010	cabo (Carsten Bormann)
require_performance.diff	28.6 KB	05/23/2011	xavershay (Xavier Shay)