Ruby trunk - Bug #6836

Improve File.expand_path performance in Windows

08/05/2012 06:55 AM - luislavena (Luis Lavena)

Status:	Closed		
Priority:	Normal		
Assignee:	luislavena (Luis Lavena)		
Target version:	2.0.0		
ruby -v:	ruby 2.0.0dev (2012-08-04 trunk 36616) [i386-mingw32]	**Backport:**	

Description

=begin
(original write up in markdown here: https://gist.github.com/3242245)

== Background

While investigating the performance issues of (({File.expand_path})) on Windows,
Usaku Nakamura and Nobuyoshi Nakada on [ruby-core:39504] pointed out that
due security concerns, accessing files on Windows required normalized paths.

This was covered in the security update of March 2008, WEBrick file-access
vulnerability [1].

After closer inspection of WEBrick code (mentioned by the security update),
I noticed it uses (({File.expand_path})) to perform the heavy lifting of path
normalization in the request.

The code around this can be inspected in (({prevent_directory_traversal}))[2]
and (({shift_path_info}))[3] methods.

This approach performs a hit into the filesystem, contrary to its
implementation in any other operating system.

(({File.expand_path})) is heavily used by (({require})), which result in slow
application startup, depending on the application size or number of gems it
depends on.

Stepping back for a second, we can see that the security issue is around
WEBrick and the way it determines (({path_info})) absoluteness.

It is also clear that to solve WEBrick security issue, a tax has been applied
to the entire Ruby ecosystem, penalizing startup performance.

With Hiroshi Shirosaki's help, we worked on a patch that:

- Limit filesystem hit only to WEBrick, using Windows' GetLongPathName [4].
- Use a Windows-specific API to normalize paths
- Improve encoding support.

What started as an experiment named Fenix [5] has shown great results on a
variety of systems.

This patch has been integrated in TheCodeShop [6] releases of Ruby 1.9.3 and
tested by Ruby-Core developers Hiroshi Shirosaki and myself.

== Performance

To demonstrate the benefits of this patch, I've used measurements [7] project
and both (({core_require_empty})) and (({core_require_nested})) workloads,
obtaining the following results:

ruby 2.0.0dev (2012-08-03 trunk 36611) [i386-mingw32]

```
Rehearsal -----------------------------------------------------
core_require_empty   1.186000   3.385000   4.571000 (  4.676267)
-------------------------------------------- total: 4.571000sec

                         user     system      total        real

core_require_empty   1.217000   3.385000   4.602000 (  4.643266)

Rehearsal ------------------------------------------------------
core_require_nested   1.514000   3.760000   5.274000 (  5.305303)
--------------------------------------------- total: 5.274000sec

                          user     system      total        real

core_require_nested   1.466000   3.713000   5.179000 (  5.233300)
```

And with patch applied:

```
ruby 2.0.0dev (2012-08-03 trunk 36611) [i386-mingw32]
Rehearsal -----------------------------------------------------
core_require_empty   0.765000   1.077000   1.842000 (  1.887603)
-------------------------------------------- total: 1.842000sec

                         user     system      total        real

core_require_empty   0.717000   1.123000   1.840000 (  1.887603)

Rehearsal ------------------------------------------------------
core_require_nested   0.717000   1.670000   2.387000 (  2.480405)
--------------------------------------------- total: 2.387000sec

                          user     system      total        real

core_require_nested   0.890000   1.528000   2.418000 (  2.496004)
```

Benchmarks were performed all on the same hardware and OS:

- CPU: Core 2 Duo T7500 @ 2.20Ghz
- RAM: 4GB
- HDD: 1.5GB RAMdisk (ImDisk)
- OS: Windows 7 Ultimate x64

All tests associated (both File and WEBrick ones) pass.

Additional tests that exercise specific aspects of new function were added.

Patch has been tested also against:

- Visual Studio build of Ruby
- Ubuntu 12.04
- Mac OSX

And the patch didn't affect either build or tests of it.

=== Real life impact: Rails

The biggest Ruby project affected by this is Rails applications.

An empty Rails application on startup requires more than 700 files from
different gems:

```
V:\emptyapp>ruby script\rails runner -e production "p $LOADED_FEATURES.size"
772
```

When benchmark startup using w32time [8]:

```
V:\emptyapp>timer ruby script\rails runner -e production "0"
```

```
real    7.285
system  4.539
user    2.683
```

And patched Ruby:

```
V:\emptyapp>timer ruby script\rails runner -e production "0"
real    2.620
system  0.873
user    1.700
```

(best result taken from several warm ups).

Now, a mid-sized application like Enki [9] which loads 1146 files in
production mode, result in:

```
V:\enki>timer ruby script\rails runner -e production "p $LOADED_FEATURES.size"
1146
real    22.620
system  11.497
user    11.076
```

Almost ((*23 seconds*)), compared to patched version:

```
V:\enki>timer ruby script\rails runner -e production "0"
real    11.013
system  1.981
user    8.938
```

This change also improves performance of (({rake})) inside Rails, from:

```
V:\enki>timer rake -T
...
real    8.689
system  0.015
user    0.000
```

To:

```
V:\enki>timer rake -T
...
real    3.307
system  0.000
user    0.031
```

Making normal operations more accessible.

Looking forward for your thoughts on these changes.

Thank you.

[1] http://www.ruby-lang.org/en/news/2008/03/03/webrick-file-access-vulnerability/
[2] https://github.com/ruby/ruby/blob/trunk/lib/webrick/httpservlet/filehandler.rb#L242-263
[3] https://github.com/ruby/ruby/blob/trunk/lib/webrick/httpservlet/filehandler.rb#L330-337
[4] http://msdn.microsoft.com/en-us/library/windows/desktop/aa364980.aspx
[5] https://github.com/luislavena/fenix
[6] http://thecodeshop.github.com/
[7] https://github.com/jonforums/measurements
[8] https://github.com/thecodeshop/w32time
[9] https://github.com/xaviershay/enki
=end

Associated revisions

Revision 86df08da - 08/24/2012 03:44 AM - luislavena (Luis Lavena)

Improve require/File.expand_path performance on Windows

- configure.in (mingw): add shlwapi to the list of dependency libs for Windows.

- win32/Makefile.sub (EXTSOLIBS): ditto.

- internal.h: declare internal functions rb_w32_init_file, rb_file_expand_path_internal and rb_file_expand_path_fast.

- file.c (Init_File): invoke Windows initialization rb_w32_init_file

- win32/file.c (rb_file_load_path_internal): new function. Windows-specific implementation that replaces file_expand_path. [Bug #6836][ruby-core:46996]

- win32/file.c (rb_w32_init_file): new function. Initialize codepage cache for faster conversion encodings lookup.

- file.c (file_expand_path): rename to rb_file_expand_path_internal. Conditionally exclude from Windows.

- file.c (rb_file_expand_path_fast): new function. delegates to rb_file_expand_path_internal without performing a hit to the filesystem.

- file.c (file_expand_path_1): use rb_file_expand_path_internal without path expansion (used by require).

- file.c (rb_find_file_ext_safe): ditto.

- file.c (rb_find_file_safe): ditto.

- load.c (rb_get_expanded_load_path): use rb_file_expand_path_fast.

- load.c (rb_feature_provided): ditto.

- file.c (rb_file_expand_path): use rb_file_expand_path_internal with path expansion.

- file.c (rb_file_absolute_path): ditto.

- test/ruby/test_file_exhaustive.rb: new tests to exercise rb_file_expand_path_internal implementation and compliance with existing behaviors.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@36811 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 36811 - 08/24/2012 03:44 AM - luislavena (Luis Lavena)

Improve require/File.expand_path performance on Windows

- configure.in (mingw): add shlwapi to the list of dependency libs for Windows.

- win32/Makefile.sub (EXTSOLIBS): ditto.

- internal.h: declare internal functions rb_w32_init_file, rb_file_expand_path_internal and rb_file_expand_path_fast.

- file.c (Init_File): invoke Windows initialization rb_w32_init_file

- win32/file.c (rb_file_load_path_internal): new function. Windows-specific implementation that replaces file_expand_path. [Bug #6836][ruby-core:46996]

- win32/file.c (rb_w32_init_file): new function. Initialize codepage cache for faster conversion encodings lookup.

- file.c (file_expand_path): rename to rb_file_expand_path_internal. Conditionally exclude from Windows.

- file.c (rb_file_expand_path_fast): new function. delegates to rb_file_expand_path_internal without performing a hit to the filesystem.

- file.c (file_expand_path_1): use rb_file_expand_path_internal without path expansion (used by require).
- file.c (rb_find_file_ext_safe): ditto.
- file.c (rb_find_file_safe): ditto.
- load.c (rb_get_expanded_load_path): use rb_file_expand_path_fast.
- load.c (rb_feature_provided): ditto.
- file.c (rb_file_expand_path): use rb_file_expand_path_internal with path expansion.
- file.c (rb_file_absolute_path): ditto.
- test/ruby/test_file_exhaustive.rb: new tests to exercise rb_file_expand_path_internal implementation and compliance with existing behaviors.

Revision 36811 - 08/24/2012 03:44 AM - luislavena (Luis Lavena)

Improve require/File.expand_path performance on Windows

- configure.in (mingw): add shlwapi to the list of dependency libs for Windows.
- win32/Makefile.sub (EXTSOLIBS): ditto.
- internal.h: declare internal functions rb_w32_init_file, rb_file_expand_path_internal and rb_file_expand_path_fast.
- file.c (Init_File): invoke Windows initialization rb_w32_init_file
- win32/file.c (rb_file_load_path_internal): new function. Windows-specific implementation that replaces file_expand_path. [Bug #6836][ruby-core:46996]
- win32/file.c (rb_w32_init_file): new function. Initialize codepage cache for faster conversion encodings lookup.
- file.c (file_expand_path): rename to rb_file_expand_path_internal. Conditionally exclude from Windows.
- file.c (rb_file_expand_path_fast): new function. delegates to rb_file_expand_path_internal without performing a hit to the filesystem.
- file.c (file_expand_path_1): use rb_file_expand_path_internal without path expansion (used by require).
- file.c (rb_find_file_ext_safe): ditto.
- file.c (rb_find_file_safe): ditto.
- load.c (rb_get_expanded_load_path): use rb_file_expand_path_fast.
- load.c (rb_feature_provided): ditto.
- file.c (rb_file_expand_path): use rb_file_expand_path_internal with path expansion.
- file.c (rb_file_absolute_path): ditto.
- test/ruby/test_file_exhaustive.rb: new tests to exercise rb_file_expand_path_internal implementation and compliance with existing behaviors.

Revision 36811 - 08/24/2012 03:44 AM - luislavena (Luis Lavena)

Improve require/File.expand_path performance on Windows

- configure.in (mingw): add shlwapi to the list of dependency libs for Windows.

- win32/Makefile.sub (EXTSOLIBS): ditto.
- internal.h: declare internal functions rb_w32_init_file, rb_file_expand_path_internal and rb_file_expand_path_fast.
- file.c (Init_File): invoke Windows initialization rb_w32_init_file
- win32/file.c (rb_file_load_path_internal): new function. Windows-specific implementation that replaces file_expand_path. [Bug #6836][ruby-core:46996]
- win32/file.c (rb_w32_init_file): new function. Initialize codepage cache for faster conversion encodings lookup.
- file.c (file_expand_path): rename to rb_file_expand_path_internal. Conditionally exclude from Windows.
- file.c (rb_file_expand_path_fast): new function. delegates to rb_file_expand_path_internal without performing a hit to the filesystem.
- file.c (file_expand_path_1): use rb_file_expand_path_internal without path expansion (used by require).
- file.c (rb_find_file_ext_safe): ditto.
- file.c (rb_find_file_safe): ditto.
- load.c (rb_get_expanded_load_path): use rb_file_expand_path_fast.
- load.c (rb_feature_provided): ditto.
- file.c (rb_file_expand_path): use rb_file_expand_path_internal with path expansion.
- file.c (rb_file_absolute_path): ditto.
- test/ruby/test_file_exhaustive.rb: new tests to exercise rb_file_expand_path_internal implementation and compliance with existing behaviors.

Revision 36811 - 08/24/2012 03:44 AM - luislavena (Luis Lavena)

Improve require/File.expand_path performance on Windows

- configure.in (mingw): add shlwapi to the list of dependency libs for Windows.

- win32/Makefile.sub (EXTSOLIBS): ditto.
- internal.h: declare internal functions rb_w32_init_file, rb_file_expand_path_internal and rb_file_expand_path_fast.
- file.c (Init_File): invoke Windows initialization rb_w32_init_file
- win32/file.c (rb_file_load_path_internal): new function. Windows-specific implementation that replaces file_expand_path. [Bug #6836][ruby-core:46996]
- win32/file.c (rb_w32_init_file): new function. Initialize codepage cache for faster conversion encodings lookup.
- file.c (file_expand_path): rename to rb_file_expand_path_internal. Conditionally exclude from Windows.
- file.c (rb_file_expand_path_fast): new function. delegates to rb_file_expand_path_internal without performing a hit to the filesystem.
- file.c (file_expand_path_1): use rb_file_expand_path_internal without

path expansion (used by require).

- file.c (rb_find_file_ext_safe): ditto.

- file.c (rb_find_file_safe): ditto.

- load.c (rb_get_expanded_load_path): use rb_file_expand_path_fast.

- load.c (rb_feature_provided): ditto.

- file.c (rb_file_expand_path): use rb_file_expand_path_internal with path expansion.

- file.c (rb_file_absolute_path): ditto.

- test/ruby/test_file_exhaustive.rb: new tests to exercise rb_file_expand_path_internal implementation and compliance with existing behaviors.

Revision 36811 - 08/24/2012 03:44 AM - luislavena (Luis Lavena)

Improve require/File.expand_path performance on Windows

- configure.in (mingw): add shlwapi to the list of dependency libs for Windows.

- win32/Makefile.sub (EXTSOLIBS): ditto.

- internal.h: declare internal functions rb_w32_init_file, rb_file_expand_path_internal and rb_file_expand_path_fast.

- file.c (Init_File): invoke Windows initialization rb_w32_init_file

- win32/file.c (rb_file_load_path_internal): new function. Windows-specific implementation that replaces file_expand_path. [Bug #6836][ruby-core:46996]

- win32/file.c (rb_w32_init_file): new function. Initialize codepage cache for faster conversion encodings lookup.

- file.c (file_expand_path): rename to rb_file_expand_path_internal. Conditionally exclude from Windows.

- file.c (rb_file_expand_path_fast): new function. delegates to rb_file_expand_path_internal without performing a hit to the filesystem.

- file.c (file_expand_path_1): use rb_file_expand_path_internal without path expansion (used by require).

- file.c (rb_find_file_ext_safe): ditto.

- file.c (rb_find_file_safe): ditto.

- load.c (rb_get_expanded_load_path): use rb_file_expand_path_fast.

- load.c (rb_feature_provided): ditto.

- file.c (rb_file_expand_path): use rb_file_expand_path_internal with path expansion.

- file.c (rb_file_absolute_path): ditto.

- test/ruby/test_file_exhaustive.rb: new tests to exercise rb_file_expand_path_internal implementation and compliance with existing behaviors.

Revision 36811 - 08/24/2012 03:44 AM - luislavena (Luis Lavena)

Improve require/File.expand_path performance on Windows

- configure.in (mingw): add shlwapi to the list of dependency libs for Windows.

- win32/Makefile.sub (EXTSOLIBS): ditto.

- internal.h: declare internal functions rb_w32_init_file, rb_file_expand_path_internal and rb_file_expand_path_fast.

- file.c (Init_File): invoke Windows initialization rb_w32_init_file

- win32/file.c (rb_file_load_path_internal): new function. Windows-specific implementation that replaces file_expand_path. [Bug #6836][ruby-core:46996]

- win32/file.c (rb_w32_init_file): new function. Initialize codepage cache for faster conversion encodings lookup.

- file.c (file_expand_path): rename to rb_file_expand_path_internal. Conditionally exclude from Windows.

- file.c (rb_file_expand_path_fast): new function. delegates to rb_file_expand_path_internal without performing a hit to the filesystem.

- file.c (file_expand_path_1): use rb_file_expand_path_internal without path expansion (used by require).

- file.c (rb_find_file_ext_safe): ditto.

- file.c (rb_find_file_safe): ditto.

- load.c (rb_get_expanded_load_path): use rb_file_expand_path_fast.

- load.c (rb_feature_provided): ditto.

- file.c (rb_file_expand_path): use rb_file_expand_path_internal with path expansion.

- file.c (rb_file_absolute_path): ditto.

- test/ruby/test_file_exhaustive.rb: new tests to exercise rb_file_expand_path_internal implementation and compliance with existing behaviors.

Revision 2c400078 - 10/25/2012 08:16 AM - usa (Usaku NAKAMURA)

merge revision(s) 34849,34853,34854,34855,34859,34862,35384,35385,36811,36812,36850,36907,36908: [Backport #7174]

```
    * Makefile.in (PLATFORM_DIR): add a variable for `win32` directory.

    * Makefile.in (clean-platform): add new target.
      It cleans `win32` directory.

    * common.mk (clean): add a dependency for `win32` directory.

    * common.mk (distclean): ditto.

    * common.mk (distclean-platform): add new target.
      It cleans `win32` directory.

    * common.mk ($(PLATFORM_D)): add new target to make `win32` directory.

    * common.mk (win32/win32.$(OBJEXT)): move win32.o into `win32`
      directory.

    * common.mk (win32/file.$(OBJEXT)): add new target for win32/file.c.

    * configure.in: move win32.o into `win32` directory and add
      win32/file.o to MISSING.

    * file.c (file_load_ok, rb_file_load_ok): replace static
      file_load_ok() with public rb_file_load_ok().
      It's to link Windows implementation in win32/file.c.
    * file.c (rb_find_file_ext_safe): ditto.

    * file.c (rb_find_file_safe): ditto.

    * win32/file.c (rb_file_load_ok): new file. Add Windows specific
      optimized implementation of rb_file_load_ok(). We created a
      separated file to avoid too many #ifdef macro which is unreadable.

    * win32/Makefile.sub (PLATFORM_DIR): add a variable for `win32`
      directory.

    * win32/Makefile.sub (MISSING): move win32.obj into `win32`
      directory and add win32/file.obj to MISSING.

    * win32/Makefile.sub (MAKEDIRS): replace MINIRUBY with BASERUBY.
      It's because miniruby doesn't exist when making `win32` directory.

    * win32/Makefile.sub (clean-platform): add new target to clean `win32`
      directory.

    * win32/Makefile.sub ({$(srcdir)}.c{}.obj): make it not match
      win32/file.c to build properly.

    * win32/Makefile.sub (win32/win32.$(OBJEXT)): move win32.obj into
     `win32` directory.
      Patch created with Luis Lavena.
      [ruby-core:42480] [Feature #5999]

    * win32/Makefile.sub (MAKEDIRS): use mkdir of cmd.exe instead of ruby.
      [Bug #6103] [ruby-core:43012]

    * win32/README.win32: added a notice about command extension of cmd.exe.

    * win32/makedirs.bat: new command to make intermediate
      directories, and not to report any errors if the directory
      already exists.

    * win32/Makefile.sub (MAKEDIRS): enable command extensions.

    * win32/file.c (INVALID_FILE_ATTRIBUTES): define for old SDK.

    * configure.in (mingw): add shlwapi to the list of dependency
      libs for Windows.

    * win32/Makefile.sub (EXTSOLIBS): ditto.

    * internal.h: declare internal functions rb_w32_init_file,
      rb_file_expand_path_internal and rb_file_expand_path_fast.

    * file.c (Init_File): invoke Windows initialization rb_w32_init_file

    * win32/file.c (rb_file_load_path_internal): new function.
      Windows-specific implementation that replaces file_expand_path.
      [Bug #6836][ruby-core:46996]

    * win32/file.c (rb_w32_init_file): new function. Initialize codepage
      cache for faster conversion encodings lookup.

    * file.c (file_expand_path): rename to rb_file_expand_path_internal.
      Conditionally exclude from Windows.

    * file.c (rb_file_expand_path_fast): new function. delegates to
      rb_file_expand_path_internal without performing a hit to the
      filesystem.

    * file.c (file_expand_path_1): use rb_file_expand_path_internal without
      path expansion (used by require).

    * file.c (rb_find_file_ext_safe): ditto.

    * file.c (rb_find_file_safe): ditto.

    * load.c (rb_get_expanded_load_path): use rb_file_expand_path_fast.
    * load.c (rb_feature_provided): ditto.

    * file.c (rb_file_expand_path): use rb_file_expand_path_internal with
      path expansion.

    * file.c (rb_file_absolute_path): ditto.

    * test/ruby/test_file_exhaustive.rb: new tests to exercise
      rb_file_expand_path_internal implementation and compliance with
      existing behaviors.

    * test/ruby/test_file_exhaustive.rb: fix test introduced in r36811 for
      posix environments where HOME is not defined.  [ruby-core:47322]
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/branches/ruby_1_9_3@37321 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 08/06/2012 12:28 PM - usa (Usaku NAKAMURA)

First, I think this is the great job!

This patch is very big, so I've not checked whole yet.
I found that this includes a patch to WEBrick.
So, I guess that this means there is an incompatibility in File.expand_path, doesn't it?
What is the incompatibility?

#2 - 08/06/2012 11:02 PM - luislavena (Luis Lavena)

usa (Usaku NAKAMURA) wrote:

> I found that this includes a patch to WEBrick.
> So, I guess that this means there is an incompatibility in File.expand_path, doesn't it?
> What is the incompatibility?

WEBrick relies on File.expand_path to resolve the traversal (by expanding) but also by expanding possible short names into long names.

Since this patch no longer hits the filesystem to determine if the path is a real file and expand the shortname into longname, we moved it to WEBrick.

From what I understand from WEBrick test, I'm still not 100% sure about that particular patch, specially since other tools like Rack handle path traversal security without touching the filesystem.

Rack doesn't rely on File.expand_path at all for traversal checks:

https://github.com/rack/rack/pull/373#issuecomment-4684709

And the code:

https://github.com/rack/rack/blob/master/lib/rack/file.rb#L40-67

Our decision to maintain WEBrick tests was under the assumption that the test was doing something other than what Rack is doing here.

Because of that, we decided to only expand the shortnames in WEBrick.

IMO think Rack approach is better, as it doesn't rely on File.expand_path at all, but I could be wrong.

#3 - 08/07/2012 09:54 AM - usa (Usaku NAKAMURA)

Hello,

Thank you for the explanation, Luis.

In message "[ruby-core:47021] [ruby-trunk - Bug #6836] Improve File.expand_path performance in Windows"
on Aug.06,2012 23:02:11, luislavena@gmail.com wrote:

> Since this patch no longer hits the filesystem to determine if the path is a real file and expand the shortname into longname, we moved it to WEBrick.

I see.

> Our decision to maintain WEBrick tests was under the assumption that the test was doing something other than what Rack is doing here.
>
> Because of that, we decided to only expand the shortnames in WEBrick.

I understand, probably :)

> IMO think Rack approach is better, as it doesn't rely on File.expand_path at all, but I could be wrong.

Normally we should not depend on the path normalization function
of File.expand_path.
Each program should perform peculiar safing processing which each
needs.
Therefore, I think that the approach of Rack may be better, too.

However, it is very difficult to write safe code because there
are too many traps in the file system of Windows.
It's impossible for non-Windows programmers in particular.
For instance, your WEBrick patch calls Win32 API with DL,
but Unix programmers will not know Win32 API.
Therefore, we made decision of pushing all the troubles in
File.expand_path.
We expected to help to write safe code in almost all cases.

Possibly this was not a good message.
Much program depending on File.expand_path may have been made
in the world.
It's the reason why I am very cowardly to change the behavior
of File.expand_path.

But I am not against to this patch.
I hope another people's review based on the above viewpoint.

Regards,
--
U.Nakamura usa@garbagecollect.jp

#4 - 08/07/2012 10:26 AM - luislavena (Luis Lavena)

On Mon, Aug 6, 2012 at 9:34 PM, U.Nakamura usa@garbagecollect.jp wrote:

> However, it is very difficult to write safe code because there
> are too many traps in the file system of Windows.
> It's impossible for non-Windows programmers in particular.
>
> For instance, your WEBrick patch calls Win32 API with DL,
> but Unix programmers will not know Win32 API.
>
> Therefore, we made decision of pushing all the troubles in
> File.expand_path.

Perhaps WEBrick patch can be changed to follow Rack approach and we
remove DL altogether.

> We expected to help to write safe code in almost all cases.

I understand, but also I noticed that File.expand_path is called all
over the place for things like require, loading files and ensuring
$LOAD_PATH is expanded (which was already expanded in the previous
require).

An empty Rails application generates 130K hits to File.expand_path
function, but not caused by Rails itself, but instead require uses it
internally.

That alone accounts for all the performance brief users of Windows
complain most of the time.

> Possibly this was not a good message.
> Much program depending on File.expand_path may have been made
> in the world.
> It's the reason why I am very cowardly to change the behavior
> of File.expand_path.
>
> But I am not against to this patch.
> I hope another people's review based on the above viewpoint.

I started working on this patch August 2011, almost a year ago.
Hiroshi and myself added the needed changes to make it work across
different encoding and versions of Windows.

Worth to mention this has been out in the wild since January, and we
had a successful user-base of people using it thanks to TheCodeShop
binary releases.

While I was thinking we could consider backport this to Ruby 1.9.3
(similar to improved IO), but maybe is time that Ruby 2.0 corrects
those assumptions.

Perhaps you can ask for feedback to the Japanese developers that use
Windows to test it out. I can definitely workout binary packages with
the patch applied if they don't want to compile themselves.

The more eyes into this will help us stop any remaining issue.

Once again, thank you for taking the time to look into this and much
appreciated your thoughts.
--
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

#5 - 08/07/2012 02:46 PM - h.shirosaki (Hiroshi Shirosaki)

Other web servers on Windows also have Windows Short (8.3) Filenames security issue.

http://www.coresecurity.com/content/filename-pseudonyms-vulnerabilities
http://www.acunetix.com/blog/web-security-zone/articles/windows-short-8-3-filenames-web-security-problem/

If a user needs to care of security for short name, perhaps disabling Windows 8.3 short name creation would be a possible solution?

Expanding short name to long name is expensive and using it frequently causes big performance difference between Windows and Unix. I think this change in 'require' and 'load' will be useful for most Windows users since not a few Windows users complain about slowness.

#6 - 08/07/2012 04:29 PM - usa (Usaku NAKAMURA)

Hello,

In message "[ruby-core:47045] [ruby-trunk - Bug #6836] Improve File.expand_path performance in Windows"
on Aug.07,2012 14:46:15, h.shirosaki@gmail.com wrote:

> Expanding short name to long name is expensive and using it frequently causes big performance difference between Windows and Unix. I think this change in 'require' and 'load' will be useful for most Windows users since not a few Windows users complain about slowness.

The Primary Principle: Security is not exchangeable for performance.

the secondary principle: but if the software is not usable by low
performance, security looses its meaning.

If the performance problem is in 'require' and 'load', change only them
and be stayed File.expand_path the same behavior.
Can't do so?

Regards,
--
U.Nakamura usa@garbagecollect.jp

#7 - 08/07/2012 08:29 PM - now (Nikolai Weibull)

On Tue, Aug 7, 2012 at 2:34 AM, U.Nakamura usa@garbagecollect.jp wrote:

> However, it is very difficult to write safe code because there
> are too many traps in the file system of Windows.
> It's impossible for non-Windows programmers in particular.
> For instance, your WEBrick patch calls Win32 API with DL,
> but Unix programmers will not know Win32 API.
> Therefore, we made decision of pushing all the troubles in

> File.expand_path.
> We expected to help to write safe code in almost all cases.

Having File.real_path instead would be nice.

#8 - 08/07/2012 10:59 PM - luislavena (Luis Lavena)

On Tue, Aug 7, 2012 at 4:28 AM, U.Nakamura usa@garbagecollect.jp wrote:

> Hello,
>
> In message "[ruby-core:47045] [ruby-trunk - Bug #6836] Improve File.expand_path performance in Windows"
> on Aug.07,2012 14:46:15, h.shirosaki@gmail.com wrote:
>
>> Expanding short name to long name is expensive and using it frequently causes big performance difference between Windows and Unix. I think this change in 'require' and 'load' will be useful for most Windows users since not a few Windows users complain about slowness.
>
> The Primary Principle: Security is not exchangeable for performance.
>
> the secondary principle: but if the software is not usable by low
> performance, security looses its meaning.

I agree with you, trading security over performance or viceversa is not good.

> If the performance problem is in 'require' and 'load', change only them
> and be stayed File.expand_path the same behavior.
> Can't do so?

Problem is expand_path is used all over the place:

load.c:
rb_get_expanded_load_path
rb_feature_provided
rb_feature_p

file.c:
rb_file_absolute_path
rb_file_identical_p
rb_file_s_absolute_path
rb_find_file_ext_safe
rb_find_file_safe

ruby.c:
expand_include_path

Changing all those places seems more complex than just changing WEBrick.

Perhaps we can make File.realpath solve that for us, after all,
realpath is supposed to resolve symlinks and perhaps could be used to
expand shortnames into longnames.

From my point of view: core security is important, so is performance.

WEBrick is stdlib, not core, changes to make stdlib happy should not
compromise core security or performance.

If shortnames (outside web) was considered a security issue for core
then I would agree that expanding shortnames needs to be in core.

I can start working on making realpath use newer Windows API (instead
of stat) to obtain the expanded filename. Perhaps that will be a nice
alternative for WEBrick.

--
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

#9 - 08/08/2012 05:33 PM - h.shirosaki (Hiroshi Shirosaki)

usa (Usaku NAKAMURA) wrote:

> If the performance problem is in 'require' and 'load', change only them
> and be stayed File.expand_path the same behavior.
> Can't do so?

I think that might be possible, though changing that properly seems not easy.
I created an experimental patch for that. This patch passed webrick test and became faster.

https://gist.github.com/3293339

Changed functions:

load.c:
rb_get_expanded_load_path
rb_feature_provided
rb_feature_p

file.c:
rb_find_file_ext_safe
rb_find_file_safe

#10 - 08/08/2012 09:42 PM - luislavena (Luis Lavena)

h.shirosaki (Hiroshi Shirosaki) wrote:

> usa (Usaku NAKAMURA) wrote:
>
> > If the performance problem is in 'require' and 'load', change only them
> > and be stayed File.expand_path the same behavior.
> > Can't do so?
>
> I think that might be possible, though changing that properly seems not easy.
> I created an experimental patch for that. This patch passed webrick test and became faster.
>
> https://gist.github.com/3293339

Hiroshi, how that compares with the original patch?

Thank you.

#11 - 08/09/2012 07:54 AM - h.shirosaki (Hiroshi Shirosaki)

On Wed, Aug 8, 2012 at 9:42 PM, luislavena (Luis Lavena)
luislavena@gmail.com wrote:

> Issue #6836 has been updated by luislavena (Luis Lavena).
>
> h.shirosaki (Hiroshi Shirosaki) wrote:
>
> > usa (Usaku NAKAMURA) wrote:
> >
> > > If the performance problem is in 'require' and 'load', change only them
> > > and be stayed File.expand_path the same behavior.
> > > Can't do so?
> >
> > I think that might be possible, though changing that properly seems not easy.
> > I created an experimental patch for that. This patch passed webrick test and became faster.
> >
> > https://gist.github.com/3293339
>
> Hiroshi, how that compares with the original patch?

Above patch is small and easy to review.
We can change the original patch to follow the above approach. We have
code for short name expansion.
I understand the original win32/file.c patch has another merit. We can
write windows specific code without too many #ifdef macro which are

hard to read.

--
Hiroshi Shirosaki

#12 - 08/23/2012 01:45 PM - luislavena (Luis Lavena)

- File improve-require-and-file-expand_path-windows.v2.diff added

=begin
usa (Usaku NAKAMURA) wrote:

> Hello,
>
> If the performance problem is in 'require' and 'load', change only them
> and be stayed File.expand_path the same behavior.
> Can't do so?

Thank you Usa for your feedback.

Based on Hiroshi experiment I've worked in an updated patch (attached)

This new patch shows the same performance boost on (({require})) without breaking backward compatibility of (({File.expand_path}))

trunk:

ruby 2.0.0dev (2012-08-23 trunk 36786) [i386-mingw32]
Rehearsal ---
core_require_empty 1.264000 3.151000 4.415000 (4.446254)
--- total: 4.415000sec

```
                      user     system      total        real
```

core_require_empty 1.154000 3.229000 4.383000 (4.432253)

Rehearsal --
core_require_nested 1.248000 3.447000 4.695000 (4.707269)
-- total: 4.695000sec

```
                       user     system      total        real
```

core_require_nested 1.467000 3.214000 4.681000 (4.699268)

patched:

```
ruby 2.0.0dev (2012-08-23 trunk 36786) [i386-mingw32]
Rehearsal -----------------------------------------------------
core_require_empty   0.593000   0.936000   1.529000 (  1.595091)
------------------------------------------- total: 1.529000sec

                         user     system      total        real
core_require_empty   0.624000   0.936000   1.560000 (  1.577090)
```

```
ruby 2.0.0dev (2012-08-23 trunk 36786) [i386-mingw32]
Rehearsal ------------------------------------------------------
core_require_nested   0.843000   0.998000   1.841000 (  1.855106)
-------------------------------------------- total: 1.841000sec

                          user     system      total        real
core_require_nested   0.764000   1.061000   1.825000 (  1.838106)
```

This also improves Rails and Rake startup time

From (trunk, mid-size Rails app):

V:\enki>timer ruby script\rails runner -e production "p $LOADED_FEATURES.size"
1135
real 22.710
system 11.013
user 11.575

V:\enki>timer rake -T
...
real 8.868

```
system  0.031
user    0.000
```

To (patched):

```
V:\enki>timer ruby script\rails runner -e production "0"
real    10.735
system  1.716
user    8.923

V:\enki>timer rake -T
...
real    3.068
system  0.015
user    0.015
```

Updated the Gist that contains the benchmark and patch:
https://gist.github.com/3242245

Considering the size of the patch and to make it more easy to review, I've pushed to my fork on GitHub the individual commits and explanation of each change here:

https://github.com/luislavena/ruby/compare/improve-require-and-file-expand_path

Looking forward for your comments and feedback.

Thank you for your time.
=end

#13 - 08/23/2012 01:46 PM - luislavena (Luis Lavena)

- Assignee changed from nobu (Nobuyoshi Nakada) to usa (Usaku NAKAMURA)

#14 - 08/23/2012 04:27 PM - usa (Usaku NAKAMURA)

- Assignee changed from usa (Usaku NAKAMURA) to luislavena (Luis Lavena)

Thanks your efforts.
Please commit. Let's test all together.

#15 - 08/24/2012 12:44 PM - luislavena (Luis Lavena)

- Status changed from Assigned to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r36811.
Luis, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

Improve require/File.expand_path performance on Windows

- configure.in (mingw): add shlwapi to the list of dependency libs for Windows.

- win32/Makefile.sub (EXTSOLIBS): ditto.

- internal.h: declare internal functions rb_w32_init_file,
  rb_file_expand_path_internal and rb_file_expand_path_fast.

- file.c (Init_File): invoke Windows initialization rb_w32_init_file

- win32/file.c (rb_file_load_path_internal): new function.
  Windows-specific implementation that replaces file_expand_path.
  [Bug #6836][ruby-core:46996]

- win32/file.c (rb_w32_init_file): new function. Initialize codepage
  cache for faster conversion encodings lookup.

- file.c (file_expand_path): rename to rb_file_expand_path_internal.
  Conditionally exclude from Windows.

- file.c (rb_file_expand_path_fast): new function. delegates to rb_file_expand_path_internal without performing a hit to the filesystem.

- file.c (file_expand_path_1): use rb_file_expand_path_internal without path expansion (used by require).
- file.c (rb_find_file_ext_safe): ditto.
- file.c (rb_find_file_safe): ditto.
- load.c (rb_get_expanded_load_path): use rb_file_expand_path_fast.
- load.c (rb_feature_provided): ditto.
- file.c (rb_file_expand_path): use rb_file_expand_path_internal with path expansion.
- file.c (rb_file_absolute_path): ditto.
- test/ruby/test_file_exhaustive.rb: new tests to exercise rb_file_expand_path_internal implementation and compliance with existing behaviors.

Files

improve-file-expand_path-windows.v1.diff	29.6 KB	08/05/2012	luislavena (Luis Lavena)
improve-require-and-file-expand_path-windows.v2.diff	32.4 KB	08/23/2012	luislavena (Luis Lavena)