Ruby master - Bug #12666

Fatal error: glibc detected an invalid stdio handle

08/10/2016 04:22 PM - vo.x (Vit Ondruch)

Status:	Open		
Priority:	Normal		
Assignee:	tenderlovemaking (Aaron Patterson)		
Target version:			
ruby -v:	ruby 2.3.1p112 (2016-04-26 revision 54768) [powerpc64-linux]	**Backport:**	2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN

Description

During build of Ruby for Fedora on PPC64, there is reported following error:

```
Fiddle::TestImport#test_io = Fatal error: glibc detected an invalid stdio handle
uncommon.mk:612: recipe for target 'yes-test-almost' failed
make: *** [yes-test-almost] Aborted (core dumped)
```

and this is the analysis of one of glibc maintainers:

```
libio vtable verification fails because there are two copies of libc.so.6 in the process:

    0x00003fffb79413a8 - 0x00003fffb7941f78 is __libc_IO_vtables in /lib64/power8/libc.so.6
    0x00003fffb74213c0 - 0x00003fffb7421f90 is __libc_IO_vtables in /lib64/libc.so.6

IO.pipe refers to a vtable from a the first copy, but the fprintf called via libffi comes from the
 second copy.

The root cause is the Fiddle module loading libc.so.6 with an absolute path:

#0  __dlopen (file=0x20728280 "/lib64/libc.so.6", mode=257) at dlopen.c:75
#1  0x00003fffb748782c in rb_fiddle_handle_initialize (argc=<optimized out>, argv=<optimized out>,
 self=544821280) at handle.c:179

This comes from test/fiddle/helper.rb:

when /x86_64-linux/
  libc_so = "/lib64/libc.so.6"
  libm_so = "/lib64/libm.so.6"
when /linux/
  libdir = '/lib'
  case [0].pack('L!').size
  when 4
    # 32-bit ruby
    libdir = '/lib32' if File.directory? '/lib32'
  when 8
    # 64-bit ruby
    libdir = '/lib64' if File.directory? '/lib64'
  end
  libc_so = File.join(libdir, "libc.so.6")
  libm_so = File.join(libdir, "libm.so.6")

So the good news is that it's just a path.  I expect the fix looks like this:

when /linux/
  libc_so = "libc.so.6"
  libm_so = "libm.so.6"

(This replaces tehe x86_64-linux branch, too.)
```

Please also note that:

```
libio vtable verification is a new security hardening feature in Fedora 25.
```

To workaround this error, I am going to apply following patch to Fedora:

```
From 346e147ba6480839b87046e9a9efab0bf6ed3660 Mon Sep 17 00:00:00 2001
From: =?UTF-8?q?V=C3=ADt=20Ondruch?= <vondruch@redhat.com>
Date: Wed, 10 Aug 2016 17:35:48 +0200
Subject: [PATCH] Rely on ldd to detect glibc.

This is just workaround, since we know we are quite sure this will be successful
on Red Hat platforms.

This workaround rhbz#1361037
---
 test/fiddle/helper.rb | 92 ----------------------------------------------
 1 file changed, 92 deletions(-)

diff --git a/test/fiddle/helper.rb b/test/fiddle/helper.rb
index 1da3d93..65148a1 100644
--- a/test/fiddle/helper.rb
+++ b/test/fiddle/helper.rb
@@ -6,98 +6,6 @@

 libc_so = libm_so = nil

-case RUBY_PLATFORM
-when /cygwin/
-  libc_so = "cygwin1.dll"
-  libm_so = "cygwin1.dll"
-when /x86_64-linux/
-  libc_so = "/lib64/libc.so.6"
-  libm_so = "/lib64/libm.so.6"
-when /linux/
-  libdir = '/lib'
-  case [0].pack('L!').size
-  when 4
-    # 32-bit ruby
-    libdir = '/lib32' if File.directory? '/lib32'
-  when 8
-    # 64-bit ruby
-    libdir = '/lib64' if File.directory? '/lib64'
-  end
-  libc_so = File.join(libdir, "libc.so.6")
-  libm_so = File.join(libdir, "libm.so.6")
-when /mingw/, /mswin/
-  require "rbconfig"
-  crtname = RbConfig::CONFIG["RUBY_SO_NAME"][/msvc\w+/] || 'ucrtbase'
-  libc_so = libm_so = "#{crtname}.dll"
-when /darwin/
-  libc_so = "/usr/lib/libc.dylib"
-  libm_so = "/usr/lib/libm.dylib"
-when /kfreebsd/
-  libc_so = "/lib/libc.so.0.1"
-  libm_so = "/lib/libm.so.1"
-when /gnu/ #GNU/Hurd
-  libc_so = "/lib/libc.so.0.3"
-  libm_so = "/lib/libm.so.6"
-when /mirbsd/
-  libc_so = "/usr/lib/libc.so.41.10"
-  libm_so = "/usr/lib/libm.so.7.0"
-when /freebsd/
-  libc_so = "/lib/libc.so.7"
-  libm_so = "/lib/libm.so.5"
-when /bsd|dragonfly/
-  libc_so = "/usr/lib/libc.so"
-  libm_so = "/usr/lib/libm.so"
-when /solaris/
-  libdir = '/lib'
-  case [0].pack('L!').size
```

```
-  when 4
-    # 32-bit ruby
-    libdir = '/lib' if File.directory? '/lib'
-  when 8
-    # 64-bit ruby
-    libdir = '/lib/64' if File.directory? '/lib/64'
-  end
-  libc_so = File.join(libdir, "libc.so")
-  libm_so = File.join(libdir, "libm.so")
-when /aix/
-  pwd=Dir.pwd
-  libc_so = libm_so = "#{pwd}/libaixdltest.so"
-  unless File.exist? libc_so
-    cobjs=%w!strcpy.o!
-    mobjs=%w!floats.o sin.o!
-    funcs=%w!sin sinf strcpy strncpy!
-    expfile='dltest.exp'
-    require 'tmpdir'
-    Dir.mktmpdir do |dir|
-      begin
-        Dir.chdir dir
-        %x!/usr/bin/ar x /usr/lib/libc.a #{cobjs.join(' ')}!
-        %x!/usr/bin/ar x /usr/lib/libm.a #{mobjs.join(' ')}!
-        %x!echo "#{funcs.join("\n")}\n" > #{expfile}!
-        require 'rbconfig'
-        if RbConfig::CONFIG["GCC"] = 'yes'
-          lflag='-Wl,'
-        else
-          lflag=''
-        end
-        flags="#{lflag}-bE:#{expfile} #{lflag}-bnoentry -lm"
-        %x!#{RbConfig::CONFIG["LDSHARED"]} -o #{libc_so} #{(cobjs+mobjs).join(' ')} #{flags}!
-      ensure
-        Dir.chdir pwd
-      end
-    end
-  end
-else
-  libc_so = ARGV[0] if ARGV[0] && ARGV[0][0] == ?/
-  libm_so = ARGV[1] if ARGV[1] && ARGV[1][0] == ?/
-  if( !(libc_so && libm_so) )
-    $stderr.puts("libc and libm not found: #{$0} <libc> <libm>")
-  end
-end
-
-libc_so = nil if !libc_so || (libc_so[0] == ?/ && !File.file?(libc_so))
-libm_so = nil if !libm_so || (libm_so[0] == ?/ && !File.file?(libm_so))
-
 if !libc_so || !libm_so
   ruby = EnvUtil.rubybin
   ldd = `ldd #{ruby}`
--
2.9.2
```

i.e. I am going to disable the custom code for detecting glibc on various platfors and rely just on ldd. My question is what should be the proper fix? Shouldn't be the ldd way the default behavior for Linux?

This issue was originally reported at:

https://bugzilla.redhat.com/show_bug.cgi?id=1361037

History

#1 - 08/12/2016 12:38 AM - shyouhei (Shyouhei Urabe)

For Fedora the patch seems OK. But I doubt if it could break on other systems like mswin. The fix proposed by the glibc maintainers seems much moderate to me.

#2 - 08/12/2016 08:45 AM - naruse (Yui NARUSE)

Shyouhei Urabe wrote:

> For Fedora the patch seems OK. But I doubt if it could break on other systems like mswin. The fix proposed by the glibc maintainers seems much moderate to me.

Use ldd instead of hard coded list sounds reasonable while ldd exists.

At least OS X doesn't have ldd(1). (it has otool -L)

#3 - 08/12/2016 09:34 AM - vo.x (Vit Ondruch)

Actually, in the RHBZ, there are two comments from two glibc maintainers and both suggest to load glibc without specifying path, e.g.:

```
when /linux/
  libc_so = "libc.so.6"
  libm_so = "libm.so.6"
```

But this should apply probably also to x86_64-linux.

What I still don't understand, why there is currently so complex logic how to determine the path, when it could be so easy?

#4 - 11/04/2016 03:57 PM - vo.x (Vit Ondruch)

Ping ... any chance to get this fixed? PPC was recently added into primary Fedora builder, so this is troublesome :/ Also, it will make issues to Fedora PPC users as soon as Fedora 25 is release, which should be next Tuesday if I am not mistaken ...

#5 - 11/04/2016 05:15 PM - kernigh (George Koehler)

- File fiddle-path.diff added

For both BSD and Linux, I want to suggest

```
when /bsd|dragonfly|linux/
  libc_so = "libc.so"
  libm_so = "libm.so"
```

Works for me on OpenBSD:

```
$ make test-all TESTS=fiddle
...
# Running tests:

Finished tests in 0.598796s, 232.1328 tests/s, 517.7061 assertions/s.
139 tests, 310 assertions, 0 failures, 0 errors, 0 skips

ruby -v: ruby 2.4.0dev (2016-11-02 trunk 56542) [x86_64-openbsd6.0]
```

It's bad to guess the path (like "/usr/lib") or the version number (like "6" in "libc.so.6"). It will break when someone has libc with different path or version number.

I'm attaching patch.

#6 - 12/12/2016 03:36 PM - naruse (Yui NARUSE)

George Koehler wrote:

> For both BSD and Linux, I want to suggest
>
> ```
> when /bsd|dragonfly|linux/
>   libc_so = "libc.so"
>   libm_so = "libm.so"
> ```

It breaks FreeBSD.

```
irb(main):001:0> require"fiddle"
=> true
irb(main):002:0> Fiddle.dlopen("libc.so")
Fiddle::DLError: /usr/lib/libc.so: invalid file format
        from /home/naruse/local/ruby/lib/ruby/2.4.0/fiddle.rb:47:in `initialize'
        from /home/naruse/local/ruby/lib/ruby/2.4.0/fiddle.rb:47:in `new'
        from /home/naruse/local/ruby/lib/ruby/2.4.0/fiddle.rb:47:in `dlopen'
```

```
        from (irb):2
        from /home/naruse/local/ruby/bin/irb:11:in `<main>'
irb(main):003:0> Fiddle.dlopen("/lib/libc.so.7")
=> #<Fiddle::Handle:0x000008031a0300>
```

Why you guys breaks other platforms without checking.

#7 - 12/21/2016 05:52 AM - shyouhei (Shyouhei Urabe)

- Assignee set to tenderlovemaking (Aaron Patterson)

- Status changed from Open to Assigned

#8 - 03/16/2020 08:12 AM - shyouhei (Shyouhei Urabe)

- Status changed from Assigned to Feedback

Sorry for a late reply, but I think we have touched this area since you reported. Does this still happen?

#9 - 04/03/2020 06:03 PM - vo.x (Vit Ondruch)

Well, we don't have PPC64 just PPC64LE on Fedora. I have run 5 builds and all passed just fine. Nevertheless, this is what glibc maintainers said about the issue 1:

> (In reply to Florian Weimer from comment #17)
>
> > libio vtable verification fails because there are two copies of libc.so.6 in
> > the process:
> >
> > 0x00003fffb79413a8 - 0x00003fffb7941f78 is __libc_IO_vtables in
> > /lib64/power8/libc.so.6
> > 0x00003fffb74213c0 - 0x00003fffb7421f90 is __libc_IO_vtables in
> > /lib64/libc.so.6
> >
> > IO.pipe refers to a vtable from a the first copy, but the fprintf called via
> > libffi comes from the second copy.
> >
> > The root cause is the Fiddle module loading libc.so.6 with an absolute path:
> >
> > #0 __dlopen (file=0x20728280 "/lib64/libc.so.6", mode=257) at dlopen.c:75
> > #1 0x00003fffb748782c in rb_fiddle_handle_initialize (argc=,
> > argv=, self=544821280) at handle.c:179
>
> This is an application defect.
>
> If you're using a path it's expected you know what you're loading.
>
> One should be using '#include ' to get LIBC_SO and then dlopen that, it's the only supported solution, particularly consider distributions >that might have /usr/lib64, or multi-arched lib dirs. You could be loading libc.so.6 from an incompatible ABI.
>
> Loding by SONAME is the only safe option.

IOW, I don't think there should be the path magic.

#10 - 04/06/2020 12:46 AM - shyouhei (Shyouhei Urabe)

- Status changed from Feedback to Closed

Thank you for confirmation.

vo.x (Vit Ondruch) wrote in #note-9:

> Well, we don't have PPC64 just PPC64LE on Fedora. I have run 5 builds and all passed just fine.

OK then, let me close this. Don't hesitate to reopen when something happens again.

> Nevertheless, this is what glibc maintainers said about the issue [1]:
> (...snip...)
>
> > This is an application defect.

Yes. However,

> If you're using a path it's expected you know what you're loading.
>
> One should be using '#include ' to get LIBC_SO and then dlopen that, it's the only supported solution, particularly consider distributions that might have /usr/lib64, or multi-arched lib dirs. You could be loading libc.so.6 from an incompatible ABI.

This doesn't work for non-glibc situations including *BSD, musl, etc.

> Loding by SONAME is the only safe option.
>
> IOW, I don't think there should be the path magic.

If the world is built on top of glibc, then yes. We can omit the entire libc detection routine. But the reality is not.

#11 - 04/06/2020 07:31 AM - vo.x (Vit Ondruch)

- Status changed from Closed to Open

- File 0001-Do-not-use-full-path-to-load-glibc.patch added

shyouhei (Shyouhei Urabe) wrote in #note-10:

> > Loding by SONAME is the only safe option.
> >
> > IOW, I don't think there should be the path magic.
>
> If the world is built on top of glibc, then yes. We can omit the entire libc detection routine. But the reality is not.

This issue is specifically about glibc branches and they should be modified. There should not be provided any path when glibc is detected. That is the problem, not detection and support of other libc implementations.

What glibc maintainers say is that there should never be loaded "/lib64/libc.so.6", it should be "libc.so.6" instead., otherwise there might end up two (different) instances of glibc loaded in memory. This, according to the glibc maintainers, is fault of Ruby and should be fixed.

IOW these two lines are wrong:

https://github.com/ruby/ruby/blob/master/test/fiddle/helper.rb#L50-L51

And they should be just:

```
libc_so = "libc.so.6"
libm_so = "libm.so.6"
```

#12 - 04/06/2020 07:32 AM - vo.x (Vit Ondruch)

Just FTR, this is from the original description:

```
0x00003fffb79413a8 - 0x00003fffb7941f78 is __libc_IO_vtables in /lib64/power8/libc.so.6
0x00003fffb74213c0 - 0x00003fffb7421f90 is __libc_IO_vtables in /lib64/libc.so.6
```

You can see that two instance of glic are loaded at the same time.

Files

fiddle-path.diff	1.37 KB	11/04/2016	kernigh (George Koehler)
0001-Do-not-use-full-path-to-load-glibc.patch	947 Bytes	04/06/2020	vo.x (Vit Ondruch)