Ruby master - Bug #16694

JIT vs hardened GCC with PCH

03/18/2020 08:30 AM - vo.x (Vit Ondruch)

Status:	Assigned		
Priority:	Normal		
Assignee:	k0kubun (Takashi Kokubun)		
Target version:			
ruby -v:	ruby 2.6.3p62 (2019-04-16 revision 67580) [x86_64-linux]	**Backport:**	2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN

Description

Preparing Ruby package for RHEL 8, I observe the following error:

~~~
$ cd /builddir/build/BUILD/ruby-2.6.3/

$ make test-all TESTS=test/ruby/test_rubyvm_mjit.rb
Run options: "--ruby=./miniruby -I./lib -I. -I.ext/common  ./tool/runruby.rb --extout=.ext  -- --d
isable-gems" --excludes-dir=./test/excludes --name=!/memory_leak/

# Running tests:

[1/4] TestRubyVMMJIT#test_pause = 0.24 s
  1) Failure:
TestRubyVMMJIT#test_pause [/builddir/build/BUILD/ruby-2.6.3/test/ruby/test_rubyvm_mjit.rb:32]:
unexpected stdout:
'''
truefalsefalse```

stderr:
'''
/tmp/_ruby_mjit_p712u0.c:1:37: error: one or more PCH files were found, but they were invalid
 #include "/tmp/_ruby_mjit_hp712u0.h"
                                     ^
compilation terminated due to -Wfatal-errors.
/tmp/_ruby_mjit_p712u1.c:1:37: error: one or more PCH files were found, but they were invalid
 #include "/tmp/_ruby_mjit_hp712u0.h"
                                     ^
compilation terminated due to -Wfatal-errors.
/tmp/_ruby_mjit_p712u2.c:1:37: error: one or more PCH files were found, but they were invalid
 #include "/tmp/_ruby_mjit_hp712u0.h"
                                     ^
compilation terminated due to -Wfatal-errors.
/tmp/_ruby_mjit_p712u3.c:1:37: error: one or more PCH files were found, but they were invalid
 #include "/tmp/_ruby_mjit_hp712u0.h"
                                     ^
compilation terminated due to -Wfatal-errors.
/tmp/_ruby_mjit_p712u4.c:1:37: error: one or more PCH files were found, but they were invalid
 #include "/tmp/_ruby_mjit_hp712u0.h"
                                     ^
compilation terminated due to -Wfatal-errors.
Successful MJIT finish
```.
<5> expected but was
<0>.

Finished tests in 0.937667s, 4.2659 tests/s, 24.5290 assertions/s.
4 tests, 23 assertions, 1 failures, 0 errors, 0 skips

ruby -v: ruby 2.6.3p62 (2019-04-16 revision 67580) [x86_64-linux]
make: *** [uncommon.mk:761: yes-test-all] Error 1
~~~

As it turns out, this is because GCC in RHEL is fully hardened. Unfortunately, due to GCC design, when GCC is fully hardened, it cannot properly handle PCH due to memory address relocation. Moreover, PCH are also security risk, so it seems they are going to be disabled entirely on RHEL.

Now I wonder what is the impact on Ruby JIT. I worry that with disabled PCH, the Ruby performance with JIT will be even worser without JIT. May be it is not good idea to use GCC for JIT. What are your thoughts?

The original ticket with all the details is here [1].

[1]: https://bugzilla.redhat.com/show_bug.cgi?id=1721553

History

#1 - 03/18/2020 08:43 AM - hsbt (Hiroshi SHIBATA)

- Assignee set to k0kubun (Takashi Kokubun)

- Status changed from Open to Assigned

#2 - 03/19/2020 03:44 AM - k0kubun (Takashi Kokubun)

> As it turns out, this is because GCC in RHEL is fully hardened. Unfortunately, due to GCC design, when GCC is fully hardened, it cannot properly handle PCH due to memory address relocation.
> it seems they are going to be disabled entirely on RHEL.

Good to know, thanks for sharing it.

> I worry that with disabled PCH, the Ruby performance with JIT will be even worser without JIT.

It's just for reducing compilation time. We should be able to support compiling it without PCH. I don't think it's a big deal.

I'll prepare a feature to disable PCH enabled by configure for RHEL.

#3 - 03/19/2020 07:25 AM - vo.x (Vit Ondruch)

k0kubun (Takashi Kokubun) wrote in #note-2:

> > I worry that with disabled PCH, the Ruby performance with JIT will be even worser without JIT.
>
> It's just for reducing compilation time. We should be able to support compiling it without PCH. I don't think it's a big deal.

But how about performance? It has to lead to slower compilation (at least the second time). I worry about usability of JIT with disabled PCH. At least it would be cool if we can document somewhere the impact, so people can decide if they want use GCC with some performance impact due to disabled PCH or used Clang with full performance.

> I'll prepare a feature to disable PCH enabled by configure for RHEL.

Thx. This configuration option would be valuable at least for testing at minimum.

#4 - 03/21/2020 07:13 PM - k0kubun (Takashi Kokubun)

> But how about performance? It has to lead to slower compilation (at least the second time). I worry about usability of JIT with disabled PCH. At least it would be cool if we can document somewhere the impact

First of all, I don't think JIT is useful for scripting purposes as most of the code may be called only a few times which leaves few optimization chances for JIT. Thus I think the main use case is a long-running server process, and performance of compiled code is more important than time taken for compilation which should be way shorter than the process's uptime.
As you said, the usability depends on how it could be slow. I agree that the impact should be documented.

> Any program creating and reading a file which somehow affects program behaviour has the same security risk. For an attacker it would be easier to corrupt some ruby source (or byte) code file loaded during CRuby work

While I didn't talk about it as I was not sure what part of PCH Vit intended is a security risk, I agree with the point for CRuby's usage if it's about a risk caused by modifying PCH. An attacker who has permission to modify Ruby's prefix should be able to modify Ruby's standard libraries too, without waiting for JIT-ing it.

> Unfortunately PCH for PIE GCC can not work with page randominazation.
> I don't see that somebody in GCC community will re-implement PCH in the same way as it is done in Clang.
>
> - use only clang for such environments

Good to know that Clang can be used as a workaround for an environment enabling page randomization.

> - header minimization (I used it originally but it does not improve JIT speed compilation when PCH is used)

Ah right... we may need it again for RHEL.

> - use one more approach based on non-fat LTO object file generated from the header as LTO works for GCC (and Clang) when page randomization (ASLR) is used.

Interesting idea. And yes, the effectiveness of inlining should be investigated before adopting this approach.

> Unfortunately, besides the advices I can not help solving this problem in the near future as I am currently busy with GCC and the light-weight JIT compiler project.

Thank you for your advice, and for all your hard work :)

#5 - 03/22/2020 02:23 AM - vmakarov (Vladimir Makarov)

On 03/21/2020 03:13 PM, takashikkbn@gmail.com wrote:

> Issue #16694 has been updated by k0kubun (Takashi Kokubun).
>
> > But how about performance? It has to lead to slower compilation (at least the second time). I worry about usability of JIT with disabled PCH. At least it would be cool if we can document somewhere the impact
> > First of all, I don't think JIT is useful for scripting purposes as most of the code may be called only a few times which leaves few optimization chances for JIT. Thus I think the main use case is a long-running server process, and performance of compiled code is more important than time taken for compilation which should be way shorter than the process's uptime.
> > As you said, the usability depends on how it could be slow. I agree that the impact should be documented.
> > Yes. You are probably right, Takashi. The major Ruby applications like
> > RoR are long running programs and you have much more experience with JIT
> > behaviour on real Ruby applications.

I just remember that for some small benchmarks, e.g. from programming
language shootout, compilation speed was important. Unfortunately,
people pay a lot of attentions to such benchmarks because it is easier
to run them.

You are right that it would be nice to know the impact of not using PH.

> > Any program creating and reading a file which somehow affects program behaviour has the same security risk. For an attacker it would be easier to corrupt some ruby source (or byte) code file loaded during CRuby work
> > While I didn't talk about it as I was not sure what part of PCH Vit intended is a security risk, I agree with the point for CRuby's usage if it's about a risk caused by modifying PCH. An attacker who has permission to modify Ruby's prefix should be able to modify Ruby's standard libraries too, without waiting for JIT-ing it.
> > Well, security people can be paranoid and sometimes not looking at the
> > whole picture. For me, because of recent security hardware
> > vulnerabilities like possibility of reading other process memory, RAM
> > seems less protected than files.
> > Unfortunately PCH for PIE GCC can not work with page randominazation.
> > I don't see that somebody in GCC community will re-implement PCH in the same way as it is done in Clang.
> >
> > - use only clang for such environments Good to know that Clang can be used as a workaround for an environment enabling page randomization. Simply GCC PH implementation is basically memory dump and to reuse it you need GCC to have the same virtual start address. GCC PH implementation is too old (I remember people started working on this more 20 years ago). So there is small chance that somebody will re-implement it.

Clang approach is real AST streaming it does not depend on Clang start
address. GCC/Clang LTO approach uses also a real IR streaming so it
works with ASLR. Coming C++ modules (GCC and Clang) also use real IR
streaming. So no danger here too.

> > - use one more approach based on non-fat LTO object file generated from the header as LTO works for GCC (and Clang) when page

randomization (ASLR) is used. Interesting idea. And yes, the effectiveness of inlining should be investigated before adopting this approach. I never investigated this. So this is just an idea. Also LTO (at least for GCC) is still pretty rarely used feature and might be a problem when people use undefined behavior (from C standard point of view) in their program.   People recently tried to use LTO for whole linux distribution and found a lot of programs with such undefined behaviour.