Ruby master - Bug #9507

Ruby 2.1.0 is broken on ARMv5: tried to create Proc object without a block

02/09/2014 03:08 PM - webmeister (_ _)

Status:	Open		
Priority:	Normal		
Assignee:	charliesome (Charlie Somerville)		
Target version:			
ruby -v:	ruby 2.1.0dev (2013-09-04 trunk 42822) [armv5tel-linux-eabi]	**Backport:**	1.9.3: UNKNOWN, 2.0.0: UNKNOWN, 2.1: UNKNOWN

Description

I'm using Arch Linux ARM which currently ships with Ruby 2.1.0. On all my ARMv5 devices, trying to run Ruby fails with a strange error, which has also been reported here: https://github.com/archlinuxarm/PKGBUILDs/issues/705 ARMv6 and ARMv7 devices on the other hand seem to be fine.

Using git-bisect I was able to determine that the first bad commit is 2f522b9cc6f3e184404040b12af4486520a73b26 (r42822), which implements #8426:

```
[root@alarm ~]# ruby --version
ruby 2.1.0dev (2013-09-04 trunk 42822) [armv5tel-linux-eabi]
[root@alarm ~]# ruby -e 'puts "foo"'
/usr/lib/ruby/2.1.0/rubygems/requirement.rb:26:in `lambda': tried to create Proc object without a block (ArgumentError)
    from /usr/lib/ruby/2.1.0/rubygems/requirement.rb:26:in `<class:Requirement>'
    from /usr/lib/ruby/2.1.0/rubygems/requirement.rb:18:in `<top (required)>'
    from /usr/lib/ruby/2.1.0/rubygems/specification.rb:10:in `require'
    from /usr/lib/ruby/2.1.0/rubygems/specification.rb:10:in `<top (required)>'
    from /usr/lib/ruby/2.1.0/rubygems.rb:1161:in `require'
    from /usr/lib/ruby/2.1.0/rubygems.rb:1161:in `<module:Gem>'
    from /usr/lib/ruby/2.1.0/rubygems.rb:114:in `<top (required)>'
    from <internal:gem_prelude>:1:in `require'
    from <internal:gem_prelude>:1:in `<compiled>'
```

The previous commit works correctly:

```
[root@alarm ~]# ruby --version
ruby 2.1.0dev (2013-09-04 trunk 42821) [armv5tel-linux-eabi]
[root@alarm ~]# ruby -e 'puts "foo"'
foo
```

The problem is still present in trunk (r44896).

History

#1 - 02/09/2014 03:37 PM - charliesome (Charlie Somerville)

Unfortunately I don't have an ARMv5 machine to debug this on. Is there anyone else who is able to take a look at this?

#2 - 02/11/2014 10:16 AM - abali (Andras Bali)

I can only confirm this bug. Could we support anyhow in debugging this?

#3 - 02/11/2014 06:07 PM - webmeister (_ _)

Do you have any idea what might be the cause here? Do you want me to try out anything?

If that helps fixing this issue, I can also give you SSH access to a SheevaPlug running Arch Linux ARM (the same system I used for investigating this issue).

#4 - 03/01/2014 04:52 PM - webmeister (_ _)

- File 0002-Use-only-unsigned-long-for-rb_serial_t.patch added

With the bug still being present in both ruby_2_1 (r45227) and trunk (r45225), I had a more detailed look at the commit that first introduced this

problem. The only somewhat architecture-dependent change in the original commit is this in internal.h:

```
+#if HAVE_UINT64_T
+    typedef uint64_t vm_state_version_t;
+#else
+    typedef unsigned long long vm_state_version_t;
+#endif
```

In the current code base the corresponding fragment reads:

```
#if defined(HAVE_LONG_LONG)
typedef unsigned LONG_LONG rb_serial_t;
#define SERIALT2NUM ULL2NUM
#elif defined(HAVE_UINT64_T)
typedef uint64_t rb_serial_t;
#define SERIALT2NUM SIZET2NUM
#else
typedef unsigned long rb_serial_t;
#define SERIALT2NUM ULONG2NUM
#endif
```

My config.h for ARMv5 contains both #define HAVE_LONG_LONG 1 and #define HAVE_UINT64_T 1, so for all versions of the code it will always take the first path. The unsigned long long variant as well as the uint64_t variant seem to be broken on ARMv5, so I decided to try the third one (unsigned long) by simply removing all other alternatives (patch attached). This successfully fixes the problem.

I'm not sure why you use the different types in the first place. If it works with unsigned long (i.e. you do not need more than 32 bits), what is the benefit of using unsigned long long? If it does bring some benefit for other architectures, could you disable its usage at least on ARMv5 or check your code whether it makes some assumptions about these types that do not hold on ARMv5?

#5 - 03/01/2014 07:33 PM - normalperson (Eric Wong)

rubylang.10.webmeister@spamgourmet.com wrote:

> My config.h for ARMv5 contains both #define HAVE_LONG_LONG 1 and
> #define HAVE_UINT64_T 1, so for all versions of the code it will
> always take the first path. The unsigned long long variant as well
> as the uint64_t variant seem to be broken on ARMv5, so I decided to
> try the third one (unsigned long) by simply removing all other
> alternatives (patch attached). This successfully fixes the problem.

Could be a compiler problem with 64-bit emulation. Which
compiler/version is this? Can you try a newer one?
Can you try any other code which uses 64-bit math?

Also, it could be an alignment problem; but I don't see places
where rb_serial_t is not 32-bit aligned...

> I'm not sure why you use the different types in the first place. If it
> works with unsigned long (i.e. you do not need more than 32 bits),
> what is the benefit of using unsigned long long? If it does bring
> some benefit for other architectures, could you disable its usage at
> least on ARMv5 or check your code whether it makes some assumptions
> about these types that do not hold on ARMv5?

64-bits is needed to avoid overflow on VM state changes.
Otherwise our caches could give false hits and crash.

#6 - 03/01/2014 08:58 PM - webmeister (_ _)

Which compiler/version is this? Can you try a newer one?

The compiler is

```
gcc (GCC) 4.8.2 20131219 (prerelease)
```

on the ARMv5 device and

```
arm-unknown-linux-gnueabi-gcc (crosstool-NG 1.19.0) 4.8.3 20131219 (prerelease)
```

for cross-compiling. GCC 4.8.2 is already the newest release.

Out of curiosity, I gave Clang 3.4 a try, but it fails already during configure:

```
checking size of int... 0
checking size of short... 0
checking size of long... 0
checking size of long long... configure: error: in `/root/clang/ruby/src/ruby-2.1.0':
configure: error: cannot compute sizeof (long long)
See `config.log' for more details
```

I'm not sure what the problem here is, Clang compiles a simple printf("%d", sizeof(long long)); just fine. Maybe some incompatibility in the configure script?

Can you try any other code which uses 64-bit math?

Can you recommend any test code? Those compilers build the whole Arch Linux ARM repository, so if it were a common problem, it should have been noticed by now.

64-bits is needed to avoid overflow on VM state changes.
Otherwise our caches could give false hits and crash.

I see. But isn't it dangerous then to fall back to unsigned long, which only guarantees at least 32 bits? Or is the probability for a crash with only 32 bits still very low, and 64 bits just provide some extra safety?

#7 - 03/01/2014 09:13 PM - normalperson (Eric Wong)

rubylang.10.webmeister@spamgourmet.com wrote:

> Can you try any other code which uses 64-bit math?
>
> Can you recommend any test code? Those compilers build the whole Arch Linux ARM repository, so if it were a common problem, it should have been noticed by now.

Maybe some crypto or audio/video codecs. It could also be compiler options for math worth trying.

> 64-bits is needed to avoid overflow on VM state changes.
> Otherwise our caches could give false hits and crash.
>
> I see. But isn't it dangerous then to fall back to unsigned long, which only guarantees at least 32 bits? Or is the probability for a crash with only 32 bits still very low, and 64 bits just provide some extra safety?

unsigned long is 64-bits on 64-bit systems. The probability of a 32-bit crash is low and only affects codebases with frequent dynamic method/module/class/const creation. Maybe few use the 32-bit unsigned long case and don't hit the problem.

#8 - 01/05/2018 09:00 PM - naruse (Yui NARUSE)

- Target version deleted (2.2.0)

Files

0002-Use-only-unsigned-long-for-rb_serial_t.patch	538 Bytes	03/01/2014	webmeister (_ _)