Ruby master - Bug #3566

memory leak when spawning+joining Threads in a loop

07/14/2010 06:55 AM - normalperson (Eric Wong)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:	1.9.2		
ruby -v:	ruby 1.9.2dev (2010-07-11 revision 28618) [x86_64-linux]	**Backport:**	

Description

=begin
The following loop causes Ruby 1.9.2-rc2 memory usage to grow without bounds:

loop { Thread.new {}.join }

I can't reproduce this with 1.9.1-p378
=end

History

#1 - 07/14/2010 08:56 PM - runpaint (Run Paint Run Run)

=begin
Confirmed on trunk: ruby 1.9.3dev (2010-07-12 trunk 28620) [i686-linux].
=end

#2 - 07/14/2010 09:43 PM - Eregon (Benoit Daloze)

=begin
On 13 July 2010 23:55, Eric Wong redmine@ruby-lang.org wrote:

> Bug #3566: memory leak when spawning+joining Threads in a loop
> http://redmine.ruby-lang.org/issues/show/3566
>
> Author: Eric Wong
> Status: Open, Priority: High
> Category: core
> ruby -v: ruby 1.9.2dev (2010-07-11 revision 28618) [x86_64-linux]
>
> The following loop causes Ruby 1.9.2-rc2 memory usage to grow without bounds:
>
> loop { Thread.new {}.join }
>
> I can't reproduce this with 1.9.1-p378

On 14 July 2010 13:56, Run Paint Run Run redmine@ruby-lang.org wrote:

> Confirmed on trunk: ruby 1.9.3dev (2010-07-12 trunk 28620) [i686-linux].

Unconfirmed on trunk, OSX: ruby 1.9.3dev (2010-07-14 trunk 28642)
[x86_64-darwin10.4.0]

Memory usage stay stable (at least after a few minutes)
At least, Thread objects get collected by GC: everytime I get 1466
Threads, GC runs and remove 50 (so 1416 left)
ruby -e 'loop { Thread.new {}.join; p ObjectSpace.each_object(Thread) {} }'

=end

#3 - 07/14/2010 10:40 PM - runpaint (Run Paint Run Run)

=begin
For me it needs to be a tight loop; your addition does indeed cause periodic reaping.
=end

#4 - 07/14/2010 11:10 PM - Eregon (Benoit Daloze)

=begin
On 14 July 2010 15:40, Run Paint Run Run redmine@ruby-lang.org wrote:

> For me it needs to be a tight loop; your addition does indeed cause periodic reaping.

I tried also without the extra code, and I do not have memory leak.

=end

#5 - 07/15/2010 05:12 PM - naruse (Yui NARUSE)

- Category set to core

- Target version set to 1.9.2

=begin

=end

#6 - 07/15/2010 10:16 PM - mame (Yusuke Endoh)

=begin
Hi,

2010/7/14 Eric Wong redmine@ruby-lang.org:

> The following loop causes Ruby 1.9.2-rc2 memory usage to grow without bounds:
>
> loop { Thread.new {}.join }

I think the following patch fixes this issue.
Even with the patch applied, memory usage seems to grow slowly.
But it will stop eventually, as long as I investigated.
I guess it is due to conservative GC.

diff --git a/thread_pthread.c b/thread_pthread.c
index e974b73..4db1226 100644
--- a/thread_pthread.c
+++ b/thread_pthread.c
@@ -213,22 +213,18 @@ get_stack(void **addr, size_t *size)
CHECK_ERR(pthread_attr_getstackaddr(&attr, addr));
CHECK_ERR(pthread_attr_getstacksize(&attr, size));
#   endif

- if (pthread_attr_getguardsize(&attr, &guard) == 0) {
- STACK_GROW_DIR_DETECTION;
- STACK_DIR_UPPER((void)0, (void)(*addr = (char *)*addr + guard));
- *size -= guard;
- } # else   CHECK_ERR(pthread_attr_init(&attr));   CHECK_ERR(pthread_attr_get_np(pthread_self(), &attr)); CHECK_ERR(pthread_attr_getstackaddr(&attr, addr));   CHECK_ERR(pthread_attr_getstacksize(&attr, size)); # endif
- CHECK_ERR(pthread_attr_getguardsize(&attr, &guard));
- *size -= guard; -# ifndef HAVE_PTHREAD_GETATTR_NP
- if (pthread_attr_getguardsize(&attr, &guard) == 0) {
- STACK_GROW_DIR_DETECTION;
- STACK_DIR_UPPER((void)0, (void)(*addr = (char *)*addr + guard));
- *size -= guard;
- }   pthread_attr_destroy(&attr); -# endif #elif defined HAVE_PTHREAD_GET_STACKADDR_NP && defined HAVE_PTHREAD_GET_STACKSIZE_NP   pthread_t th = pthread_self();   *addr = pthread_get_stackaddr_np(th);

--
Yusuke Endoh mame@tsg.ne.jp

=end

#7 - 07/16/2010 04:37 AM - normalperson (Eric Wong)

=begin
Yusuke ENDOH mame@tsg.ne.jp wrote:

> 2010/7/14 Eric Wong redmine@ruby-lang.org:

> > > The following loop causes Ruby 1.9.2-rc2 memory usage to grow without bounds:
> > >
> > > loop { Thread.new {}.join }
> >
> > I think the following patch fixes this issue.
> > Even with the patch applied, memory usage seems to grow slowly.
> > But it will stop eventually, as long as I investigated.
> > I guess it is due to conservative GC.

Yes, this helps stabilize memory growth. I'll let it run for a few
hours here and report back if I OOM my machine :)

The other weird thing is this loop takes 118M RSS with 1.9.1-rc2, but
1.9.1-p378 only takes 13M. This is a huge memory difference.

I'm running x86_64 Linux

--
Eric Wong

=end

#8 - 07/16/2010 05:41 AM - normalperson (Eric Wong)

=begin
Eric Wong normalperson@yhbt.net wrote:

> Yusuke ENDOH mame@tsg.ne.jp wrote:
>
> > 2010/7/14 Eric Wong redmine@ruby-lang.org:
> >
> > > The following loop causes Ruby 1.9.2-rc2 memory usage to grow without bounds:
> > >
> > > loop { Thread.new {}.join }
> >
> > I think the following patch fixes this issue.
> > Even with the patch applied, memory usage seems to grow slowly.
> > But it will stop eventually, as long as I investigated.
> > I guess it is due to conservative GC.
>
> Yes, this helps stabilize memory growth. I'll let it run for a few
> hours here and report back if I OOM my machine :)

Still stable (at 118M) after an hour \o/

> The other weird thing is this loop takes 118M RSS with 1.9.1-rc2, but
> 1.9.1-p378 only takes 13M. This is a huge memory difference.

Looks like GC changed between 1.9.1 and 1.9.2, throwing some random code
in there lowers memory usage and in real-ish-world case of Rainbows!
"hello world" app and ThreadSpawn concurrency, they use around the same
RSS. Sorry for the noise.

--
Eric Wong

=end

#9 - 07/16/2010 08:42 PM - mame (Yusuke Endoh)

=begin
Hi,

2010/7/16 Eric Wong normalperson@yhbt.net:

> Eric Wong normalperson@yhbt.net wrote:
>
> > Yusuke ENDOH mame@tsg.ne.jp wrote:

> > > 2010/7/14 Eric Wong redmine@ruby-lang.org:
> > >
> > > > The following loop causes Ruby 1.9.2-rc2 memory usage to grow without bounds:
> > > >
> > > > loop { Thread.new {}.join }
> > >
> > > I think the following patch fixes this issue.
> > > Even with the patch applied, memory usage seems to grow slowly.
> > > But it will stop eventually, as long as I investigated.
> > > I guess it is due to conservative GC.
> >
> > Yes, this helps stabilize memory growth. I'll let it run for a few
> > hours here and report back if I OOM my machine :)
>
> Still stable (at 118M) after an hour \o/

Thanks!

I'll commit the following patch instead of the previous one, because
the previous one seems to cause SEGV when running make test. (sorry!)

```
diff --git a/thread_pthread.c b/thread_pthread.c
index e974b73..e832b82 100644
--- a/thread_pthread.c
+++ b/thread_pthread.c
@@ -226,9 +226,7 @@ get_stack(void **addr, size_t *size)
# endif
CHECK_ERR(pthread_attr_getguardsize(&attr, &guard));
*size -= guard;
-# ifndef HAVE_PTHREAD_GETATTR_NP
pthread_attr_destroy(&attr);
-# endif
#elif defined HAVE_PTHREAD_GET_STACKADDR_NP && defined
HAVE_PTHREAD_GET_STACKSIZE_NP
pthread_t th = pthread_self();
*addr = pthread_get_stackaddr_np(th);
```

--
Yusuke Endoh mame@tsg.ne.jp

=end

#10 - 07/16/2010 10:28 PM - mame (Yusuke Endoh)

=begin
2010/7/16 Rocky Bernstein rocky.bernstein@gmail.com:

> I am now getting a SEGV presumably in a garbage collection routine (gc_mark)
> on trunk (SVN revision 28656), but about 15 hours ago I didn't. I don't get
> the crash every time I run "make check" but in 1 out of 3 tries I do.
> Attached should be a slightly stripped down log of a "make check" run from
> Ubuntu.

I've not committed my patch yet.
Do you mean that you did apply it and then got SEGV?
Otherwise, could you revert r28656 and check if SEGV occurs or not?

--
Yusuke Endoh mame@tsg.ne.jp

=end

#11 - 07/17/2010 05:45 AM - normalperson (Eric Wong)

=begin
Yusuke ENDOH mame@tsg.ne.jp wrote:

> I'll commit the following patch instead of the previous one, because
> the previous one seems to cause SEGV when running make test. (sorry!)

```
diff --git a/thread_pthread.c b/thread_pthread.c
index e974b73..e832b82 100644
--- a/thread_pthread.c
+++ b/thread_pthread.c
@@ -226,9 +226,7 @@ get_stack(void **addr, size_t *size)
# endif
CHECK_ERR(pthread_attr_getguardsize(&attr, &guard));
*size -= guard;
-# ifndef HAVE_PTHREAD_GETATTR_NP
pthread_attr_destroy(&attr);
-# endif
#elif defined HAVE_PTHREAD_GET_STACKADDR_NP && defined
HAVE_PTHREAD_GET_STACKSIZE_NP
pthread_t th = pthread_self();
*addr = pthread_get_stackaddr_np(th);
```

Thanks Yusuke, this patch works great for me and is much easier to
understand :)   #ifdef blocks inside functions scare me :x

--
Eric Wong

=end

#12 - 07/22/2010 08:17 PM - mame (Yusuke Endoh)

- Status changed from Open to Closed

- % Done changed from 0 to 100

=begin
This issue was solved with changeset r28716.
Eric, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

=end