Ruby - Feature #13637

[PATCH] tool/runruby.rb: test with smallest possible machine stack

06/06/2017 11:02 PM - normalperson (Eric Wong)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

Lets ensure none of our C functions use too much stack space and
fix all excessive stack usage before releasing the next version.
Reducing C stack usage should reduce conservative GC scanning
time and improve performance.

Hopefully there are no objections; I will commit in a few days.

If there are platform-dependent CI failures; excessive stack
usage should be fixed; rather than increasing minimum values or
removing these envs from testing.

Related issues:

Related to Ruby - Bug #13757: TestBacktrace#test_caller_lev segaults on PPC	**Closed**

Associated revisions

Revision c4e2cf466448f4283fd3f8a17a73f5fa9b745fe1 - 06/08/2017 08:58 PM - Eric Wong

tool/runruby.rb: test with smallest possible machine stack

Lets ensure none of our C functions use too much stack space and
fix all excessive stack usage before releasing the next version.
Reducing C stack usage should reduce conservative GC scanning
time and improve performance.

If there are platform-dependent test failures; excessive stack
usage should be fixed; rather than increasing minimum values or
removing these envs from testing.

- tool/runruby.rb: use smallest possible machine stack size
  [ruby-core:81597] [Feature #13637]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@59047 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision c4e2cf46 - 06/08/2017 08:58 PM - Eric Wong

tool/runruby.rb: test with smallest possible machine stack

Lets ensure none of our C functions use too much stack space and
fix all excessive stack usage before releasing the next version.
Reducing C stack usage should reduce conservative GC scanning
time and improve performance.

If there are platform-dependent test failures; excessive stack
usage should be fixed; rather than increasing minimum values or
removing these envs from testing.

- tool/runruby.rb: use smallest possible machine stack size
  [ruby-core:81597] [Feature #13637]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@59047 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 3215b27a9abd8de793cf517f32d8901fd421eb1c - 07/28/2017 04:47 PM - Rei Odaira

Include sys/select.h when checking HAVE_RB_FD_INIT

- configure.in: include sys/select.h for fd_mask on AIX
  [Feature #13637]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@59440 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 3215b27a - 07/28/2017 04:47 PM - Rei Odaira

Include sys/select.h when checking HAVE_RB_FD_INIT

- configure.in: include sys/select.h for fd_mask on AIX
  [Feature #13637]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@59440 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 06/08/2017 08:58 PM - Anonymous

- Status changed from Open to Closed

Applied in changeset trunk|r59047.

---

tool/runruby.rb: test with smallest possible machine stack

Lets ensure none of our C functions use too much stack space and
fix all excessive stack usage before releasing the next version.
Reducing C stack usage should reduce conservative GC scanning
time and improve performance.

If there are platform-dependent test failures; excessive stack
usage should be fixed; rather than increasing minimum values or
removing these envs from testing.

- tool/runruby.rb: use smallest possible machine stack size
  [ruby-core:81597] [Feature #13637]

#2 - 06/09/2017 04:35 AM - ko1 (Koichi Sasada)

I missed this ticket.
I wonder there are no failures on CI.

Do you mean that we shouldn't use recursive call which can increase machine stack on Thread and Fiber in our tests?
Now, we don't have such tests (so that we don't have failures/errors) but it is possible.

I agree that we should consider about machine stack size, but I'm not sure this approach is correct (at least now it seems no problem). Or if we need to introduce such recursive calls, we remove this restriction?

Thanks,
Koichi

#3 - 06/09/2017 07:41 AM - normalperson (Eric Wong)

ko1@atdot.net wrote:

> I missed this ticket.
> I wonder there are no failures on CI.

That is fortunate to hear :)

> Do you mean that we shouldn't use recursive call which can increase machine stack on Thread and Fiber in our tests?
> Now, we don't have such tests (so that we don't have failures/errors) but it is possible.

We should reconsider our code and data structures before
introducing recursion in code we control.

> I agree that we should consider about machine stack size, but I'm not sure this approach is correct (at least now it seems no problem). Or if we need to introduce such recursive calls, we remove this restriction?

If we really need to introduce recursion; we can use
assert_separately to test it.

However, we should avoid recursion if possible because of GC
cost and potential portability + safety problems. Instead; we
can redesign data structures and algorithms to avoid recursion

(and maybe encourage Rubyists to do the same).

Coincidentally, I (finally) announced msgthr earlier on ruby-talk;
which makes non-recursive modifications to previously well-known
recursive algorithm:

http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/437984

These were originally implemented in Perl5:

Mail::Thread in CPAN:

https://rt.cpan.org/Ticket/Display.html?id=116727
http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=833479

and public-inbox:

https://public-inbox.org/meta/20160621031201.28089-1-e@80x24.org/t/

#4 - 07/24/2017 08:28 PM - ReiOdaira (Rei Odaira)

Ruby CI on AIX have frequently hit SystemStackError since this change was introduced.
http://rubyci.s3.amazonaws.com/aix71_ppc/ruby-trunk/recent.html
http://rubyci.s3.amazonaws.com/aix71_ppc/ruby-trunk/log/20170723T103301Z.fail.html.gz

> If there are platform-dependent test failures; excessive stack usage should be fixed; rather than increasing minimum values or removing these envs from testing.

How do you think we can fix the "excessive stack usage"?

#5 - 07/24/2017 09:08 PM - normalperson (Eric Wong)

Rei.Odaira@gmail.com wrote:

> Ruby CI on AIX have frequently hit SystemStackError since this change was introduced.
> http://rubyci.s3.amazonaws.com/aix71_ppc/ruby-trunk/recent.html
> http://rubyci.s3.amazonaws.com/aix71_ppc/ruby-trunk/log/20170723T103301Z.fail.html.gz

Either that is the 16K buffer in IO.copy_stream (see below) or
OpenSSL itself is using lots of stack. I don't think we can
fix OpenSSL... (curious, which version do you use?)

>> If there are platform-dependent test failures; excessive stack usage should be fixed; rather than increasing minimum values or removing these envs from testing.
>
> How do you think we can fix the "excessive stack usage"?

Does Linux checkstack.pl work for your binaries?

https://80x24.org/mirrors/linux.git/plain/scripts/checkstack.pl
(usage in comment)

IO.copy_stream buffer:

Can you try the following patch to move allocation from stack
to heap? It may slow down small copies a little, but releasing
GVL also hurts, so I doubt the slow down will be noticeable.

diff --git a/io.c b/io.c
index 60af120c18..f4b3fcec4a 100644
--- a/io.c
+++ b/io.c
@@ -10692,7 +10692,7 @@ nogvl_copy_stream_write(struct copy_stream_struct *stp, char *buf, size_t len)
static void
nogvl_copy_stream_read_write(struct copy_stream_struct *stp)
{

- char buf[1024*16];

- char *buf;
  size_t len;

ssize_t ss;
int ret;
@@ -10702,6 +10702,13 @@ nogvl_copy_stream_read_write(struct copy_stream_struct *stp)
int use_pread;

copy_length = stp->copy_length;

- if (copy_length < 0) {
- buf = xmalloc(16384);
- }
- else {
- buf = xmalloc(copy_length > 16384 ? 16384 : copy_length);
- }

use_eof = copy_length == (off_t)-1;
src_offset = stp->src_offset;
use_pread = src_offset != (off_t)-1;

Thanks

#6 - 07/25/2017 02:51 AM - normalperson (Eric Wong)

Sorry, original patch was broken :x (yet "make exam" passed...)
(it leaked memory and used sizeof improperly)

Can you try the following, instead?

diff --git a/io.c b/io.c
index 60af120c18..0d5ca0d95b 100644
--- a/io.c
+++ b/io.c
@@ -10692,7 +10692,7 @@ nogvl_copy_stream_write(struct copy_stream_struct *stp, char *buf, size_t len)
static void
nogvl_copy_stream_read_write(struct copy_stream_struct *stp)
{

- char buf[1024*16];

- char *buf;
  size_t len;
  ssize_t ss;
  int ret;
  @@ -10700,8 +10700,14 @@ nogvl_copy_stream_read_write(struct copy_stream_struct *stp)
  int use_eof;
  off_t src_offset;
  int use_pread;
- size_t alloc_size = 16384;

copy_length = stp->copy_length;

- if (copy_length > 0 && copy_length < alloc_size) {
- alloc_size = copy_length;
- }
- buf = xmalloc(alloc_size);

use_eof = copy_length == (off_t)-1;
src_offset = stp->src_offset;
use_pread = src_offset != (off_t)-1;
@@ -10713,18 +10719,18 @@ nogvl_copy_stream_read_write(struct copy_stream_struct *stp)
if (r == (off_t)-1 && errno) {
stp->syserr = "lseek";
stp->error_no = errno;

- [redacted]

- [redacted]

}
src_offset = (off_t)-1;
use_pread = 0;
}

while (use_eof || 0 < copy_length) {

- [redacted]

- [redacted]

len = (size_t)copy_length;
}
else {

- [redacted]

- [redacted]

}
if (use_pread) {
ss = maygvl_copy_stream_read(0, stp, buf, len, src_offset);
@@ -10735,15 +10741,17 @@ nogvl_copy_stream_read_write(struct copy_stream_struct *stp)*
ss = maygvl_copy_stream_read(0, stp, buf, len, (off_t)-1);
}
if (ss <= 0) / EOF or error */

- [redacted]

- [redacted]

ret = nogvl_copy_stream_write(stp, buf, ss);
if (ret < 0)

- [redacted]

- [redacted]

if (!use_eof)
copy_length -= ss;
}
+out:

- free(buf);
  }

static void *

Thanks.

#7 - 07/26/2017 08:40 PM - ReiOdaira (Rei Odaira)

Thanks for the patch. Unfortunately, it did not solve the problem. Looks like this test does not call nogvl_copy_stream_read_write() but instead calls copy_stream_fallback_body(). As far as I read the code, there is no large array stack-allocated on that path...?

#8 - 07/27/2017 09:41 AM - normalperson (Eric Wong)

Rei.Odaira@gmail.com wrote:

> Thanks for the patch. Unfortunately, it did not solve the problem. Looks like this test does not call nogvl_copy_stream_read_write() but instead calls copy_stream_fallback_body(). As far as I read the code, there is no large array stack-allocated on that path...?

Ah, looks like you're right. Hmm.. which OpenSSL version do you
use? Perhaps we can set a higher stack size for some versions
of OpenSSL on AIX; I don't think we've seen this other
platforms...

Also, are NFDBITS and HAVE_RB_FD_INIT macros defined?
Platforms without them will allocate select() bitmaps on stack;
which can get big.

Perhaps enabling the (currently Linux-only) poll()
rb_wait_for_single_fd can avoid big bitmaps for you:

diff --git a/thread.c b/thread.c
index b7ee1d8d9b..c9e52b8698 100644
--- a/thread.c
+++ b/thread.c
@@ -3823,7 +3823,7 @@ rb_thread_fd_select(int max, rb_fdset_t * read, rb_fdset_t * write, rb_fdset_t *

- one we know of that supports using poll() in all places select()
- would work.
  */
  -#if defined(HAVE_POLL) && defined(**linux**)
  +#if defined(HAVE_POLL) && (defined(**linux**) || defined(_AIX))

define USE_POLL

#endif

Also, does pahole work on your binaries?
git clone git://git.kernel.org/pub/scm/devel/pahole/pahole.git

That helps find down big stack users (including OpenSSL
or any other 3rd party binaries).

#9 - 07/28/2017 04:57 PM - ReiOdaira (Rei Odaira)

I think I am using openssl-1.0.1s.

HAVE_RB_FD_INIT is not defined, but in fact AIX has fd_mask. It turned out that when defining HAVE_RB_FD_INIT by checking fd_mask, configure.in does not include sys/select.h. On AIX you have to explicitly include it to have fd_mask. I fixed it in r59440, and the SystemStackError disappeared.

Thanks much for your help!

#10 - 08/25/2017 11:45 AM - vo.x (Vit Ondruch)

- Related to Bug #13757: TestBacktrace#test_caller_lev segaults on PPC added

Files

0001-tool-runruby.rb-test-with-smallest-possible-machine-.patch	1.09 KB	06/06/2017	normalperson (Eric Wong)