Ruby master - Feature #2294

[PATCH] ruby_bind_stack() to embed Ruby in coroutine

10/28/2009 02:02 AM - sunaku (Suraj Kurapati)

Status:	Assigned
Priority:	Normal
Assignee:	ko1 (Koichi Sasada)
Target version:	

Description

=begin
Hi,

I am attaching a "ruby_bind_stack.patch" patch file
that adds a ruby_bind_stack() function to the Ruby C API.

This function allows me to inform the GC about the stack
boundaries of the coroutine inside which Ruby is embedded:

void ruby_bind_stack(void *lower, void *upper);

I am also attaching tarballs containing code examples that
embed Ruby inside two different coroutine environments:
UNIX System V contexts1 and libpcl2 coroutines.

Each tarball has an "output.log" file which contains the
result of running script -c ./run.sh output.log on my
machine:

Linux yantram 2.6.31-ARCH #1 SMP PREEMPT Tue Oct 13 13:36:23 CEST 2009 i686 Intel(R) Pentium(R) D CPU 3.00GHz GenuineIntel GNU/Linux

The last section in "output.log" corresponds to Ruby @ SVN
trunk that is patched with the "ruby_bind_stack.patch"
patch file that is attached to this issue.

Thanks for your consideration.

See also:

- http://redmine.ruby-lang.org/issues/show/2258
- http://redmine.ruby-lang.org/issues/show/2126 =end

Related issues:

Is duplicate of Ruby master - Feature #2126: ruby_init_stack() - add ability ...	**Closed**	**09/20/2009**
Is duplicate of Ruby master - Bug #2258: Kernel#require inside rb_require() i...	**Closed**	**10/23/2009**

Associated revisions

Revision e5481ccd - 01/23/2013 04:39 AM - ko1 (Koichi Sasada)

- thread_pthread.c (ruby_init_stack): ignore `STACK_END_ADDRESS' if Ruby interpreter is running on co-routine. [Feature #2294] https://bugs.ruby-lang.org/issues/2294#note-18

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@38905 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 38905 - 01/23/2013 04:39 AM - ko1 (Koichi Sasada)

- thread_pthread.c (ruby_init_stack): ignore `STACK_END_ADDRESS' if Ruby interpreter is running on co-routine. [Feature #2294] https://bugs.ruby-lang.org/issues/2294#note-18

Revision 38905 - 01/23/2013 04:39 AM - ko1 (Koichi Sasada)

- thread_pthread.c (ruby_init_stack): ignore `STACK_END_ADDRESS' if Ruby interpreter is running on co-routine. [Feature #2294] https://bugs.ruby-lang.org/issues/2294#note-18

Revision 38905 - 01/23/2013 04:39 AM - ko1 (Koichi Sasada)

- thread_pthread.c (ruby_init_stack): ignore `STACK_END_ADDRESS' if Ruby interpreter is running on co-routine. [Feature #2294] https://bugs.ruby-lang.org/issues/2294#note-18

Revision 38905 - 01/23/2013 04:39 AM - ko1 (Koichi Sasada)

- thread_pthread.c (ruby_init_stack): ignore `STACK_END_ADDRESS' if Ruby interpreter is running on co-routine. [Feature #2294] https://bugs.ruby-lang.org/issues/2294#note-18

Revision 38905 - 01/23/2013 04:39 AM - ko1 (Koichi Sasada)

- thread_pthread.c (ruby_init_stack): ignore `STACK_END_ADDRESS' if Ruby interpreter is running on co-routine. [Feature #2294] https://bugs.ruby-lang.org/issues/2294#note-18

Revision 38905 - 01/23/2013 04:39 AM - ko1 (Koichi Sasada)

- thread_pthread.c (ruby_init_stack): ignore `STACK_END_ADDRESS' if Ruby interpreter is running on co-routine. [Feature #2294] https://bugs.ruby-lang.org/issues/2294#note-18

History

#1 - 10/28/2009 02:33 AM - sunaku (Suraj Kurapati)

- File ruby_bind_stack.patch added

=begin

=end

#2 - 10/28/2009 09:36 AM - nobu (Nobuyoshi Nakada)

=begin
Hi,

At Wed, 28 Oct 2009 02:03:01 +0900,
Suraj Kurapati wrote in [ruby-core:26361]:

> I am attaching a "ruby_bind_stack.patch" patch file
> that adds a ruby_bind_stack() function to the Ruby C API.

This patch does not work with multithreading at all.

--
Nobu Nakada

=end

#3 - 10/28/2009 01:22 PM - sunaku (Suraj Kurapati)

- File ruby_bind_stack.patch added

=begin
Hi,

Nobu Nakada wrote:

> This patch does not work with multithreading at all.

Thank you for pointing out this problem. I have updated
my patch accordingly and am reattaching it to this issue.

Here is my approach for solving this problem:
(Please correct me if I am wrong.)

Since Ruby 1.9 threads are native kernel threads, they

dynamically allocate and manage their own stacks. So
the ruby_bind_stack() GC marking restriction must only
be applied to the main Ruby thread---which isn't really
a thread at all; it runs on the native C program stack.

Thanks for your consideration.
=end

#4 - 10/28/2009 03:02 PM - sunaku (Suraj Kurapati)

- File ruby-ucontext-thread.tgz added

- File ruby-libpcl-thread.tgz added

=begin
Hi,

I am attaching two updated code examples which
test the multi-threading support of my updated
"ruby_bind_stack.patch" patch file.

One example uses UNIX System V contexts and the
other uses libpcl for embedding Ruby in coroutine.

Thanks for your consideration.
=end

#5 - 10/28/2009 07:04 PM - romanbsd (Roman Shterenzon)

=begin
I'm embedding a Ruby 1.9.1 in my app, and it would die with segmentation fault, and I was suspicious about the stack, as it's multithreaded. I applied your patch and it looks fine so far. I'm using the following code (and assume that stack grows upward):

static void pthread_get_stack(void **stack_begin, void **stack_end) {*
size_t stack_size;
#if defined(HAVE_STACKADDR_NP) && defined(HAVE_GET_STACKSIZE_NP) / MacOS X */*
pthread_t t_id = pthread_self();
**stack_begin = pthread_get_stackaddr_np(t_id);*
stack_size = pthread_get_stacksize_np(t_id);
#else / Linux */*
pthread_attr_t attr;
pthread_getattr_np(pthread_self(), &attr);
pthread_attr_getstack(&attr, stack_begin, &stack_size);
#endif
*stack_end = *stack_begin + stack_size;
}

=end

#6 - 10/28/2009 10:51 PM - sunaku (Suraj Kurapati)

- File ruby_bind_stack.patch added

=begin

=end

#7 - 10/29/2009 12:37 AM - sunaku (Suraj Kurapati)

=begin
Hi Roman,

I did not understand your code example.
Where do you call ruby_bind_stack() ?

Without that call, I don't see how my patch
can make any difference to your program.

Thanks.
=end

#8 - 10/29/2009 06:01 AM - romanbsd (Roman Shterenzon)

=begin
Sorry for the lack of explanation, I thought that it was implicit and apparent.

After I get thread's stack_begin and stack_end I'm calling your bind stack function, of course.
=end

#9 - 10/29/2009 06:46 AM - sunaku (Suraj Kurapati)

=begin
Roman, thanks for clarifying.
=end

#10 - 10/30/2009 05:13 AM - sunaku (Suraj Kurapati)

- File ruby_bind_stack.patch added

=begin
Hi,

I am attaching an updated "ruby_bind_stack.patch" file which adds:

- API documentation for the ruby_bind_stack() function in ruby.h
- an assertion to ensure that upper > lower inside ruby_bind_stack()

Thanks for your consideration.
=end

#11 - 10/30/2009 05:37 AM - sunaku (Suraj Kurapati)

- File ruby_bind_stack.patch added

=begin

=end

#12 - 10/30/2009 05:39 AM - sunaku (Suraj Kurapati)

- File ruby_bind_stack.patch added

=begin

=end

#13 - 10/30/2009 03:55 PM - sunaku (Suraj Kurapati)

- File ruby_bind_stack.patch added

=begin

=end

#14 - 10/31/2009 02:25 PM - sunaku (Suraj Kurapati)

- File ruby_bind_stack_after_refactoring.patch added

=begin
Hi,

To reduce your risk of applying (or even considering) this patch,
I moved the refactoring of *duplicated* machine stack calculation
code into a new "get_machine_stack_bounds.patch" file on this issue:

http://redmine.ruby-lang.org/issues/show/2315

I am attaching a new "ruby_bind_stack_after_refactoring.patch" which
basically contains the result of "ruby_bind_stack.patch" *minus* the
changes in the "get_machine_stack_bounds.patch" mentioned above.

Thanks for your consideration.
=end

#15 - 11/01/2009 03:32 AM - sunaku (Suraj Kurapati)

- File ruby_bind_stack.patch added

- File ruby_bind_stack_after_refactoring.patch added

=begin
Hi,

I'm attaching updated patches that contain better API documentation:

/*

- Binds the stack of Ruby's main thread to the region of memory that spans
- inclusively from the given lower boundary to the given upper boundary: *
- lower boundary <= stack pointer of Ruby's main thread <= upper boundary *
- These boundaries *do not* protect Ruby's main thread against stack
- overflow and they *do not* apply to non-main Ruby threads (whose stacks
- are dynamically allocated and managed by the native Operating System). */ void ruby_bind_stack(void *lower_boundary, void *upper_boundary);

Thanks for your consideration.
=end

#16 - 11/01/2009 04:12 AM - sunaku (Suraj Kurapati)

- File ruby_bind_stack.patch added

- File ruby_bind_stack_after_refactoring.patch added

=begin
Hi,

I'm attaching updated patches that reduce the
runtime overhead of stack bound correction.

Thanks for your consideration.
=end

#17 - 11/04/2009 02:28 AM - sunaku (Suraj Kurapati)

- File ruby_bind_stack_r25604.patch added

=begin
Hi,

Since my refactoring patch (from issue #2315) was accepted in r25604,
I am attaching a new "ruby_bind_stack_r25604.patch" file accordingly.

Thanks for your consideration.

=end

#18 - 11/05/2009 04:27 PM - sunaku (Suraj Kurapati)

=begin
Hi,

According to Matz's suggestion in [ruby-core:25139], I wrote
a detailed explanation of the problem this patch solves. I
hope this explanation is helpful. Please do not hesitate
to ask for clarifications or to correct any misunderstandings.

Thanks for your thoughtful consideration.

== Introduction

The patch adds a ruby_bind_stack() function to the Ruby C API.
This function allows the person who is embedding Ruby to
tell the Ruby GC about the stack boundaries of the embedded
environment:

void ruby_bind_stack(VALUE *lower_bound, VALUE *upper_bound);

In order to understand why this function is important, please
consider the following two modes of operation: normal & embedded.

== Normal operation: Ruby runs in a C program's main()

Initially, Ruby assumes that the stack of Ruby's main
thread exists in a high memory address range, like this:

(high memory address)

0xc1bff1f0 Ruby's stack upper boundary

0xbffff1f0 Ruby's stack lower boundary

(low memory address)

As Ruby runs, the lower boundary is adjusted (by the
SET_STACK_END macro) to reflect the machine stack pointer:

(high memory address)

0xc1bff1f0 Ruby's stack upper boundary (not changed)

0xc0ff1e80 Ruby's stack lower boundary
(after update by SET_STACK_END)

(low memory address)

== Embedded operation: Ruby runs inside a C coroutine

Initially, Ruby assumes that the stack of Ruby's main
thread exists in a high memory address range, like this:

(high memory address)

0xc1bff1f0 Ruby's stack upper boundary

0xbffff1f0 Ruby's stack lower boundary

(low memory address)

However, the stack of the C coroutine (which runs Ruby)
exists at a low memory address range, because it is
statically allocated:

(high memory address)

0xc1bff1f0 Ruby's stack upper boundary

0xbffff1f0 Ruby's stack lower boundary

0x086032a0 System V context's stack upper boundary

0x082032a0 System V context's stack lower boundary

(low memory address)

As Ruby runs, the lower boundary is adjusted (by the
SET_STACK_END macro) to reflect the machine stack pointer:

(high memory address)

0xc1bff1f0 Ruby's stack upper boundary

0x086032a0 System V context's stack upper boundary

0x08601680 Ruby's stack lower boundary
(after update by SET_STACK_END)

0x082032a0 System V context's stack lower boundary

(low memory address)

See the problem? Ruby's stack and the C coroutine stack
do not agree. They overlap!

This situation becomes worse (and causes a segfault) when
the Ruby GC runs: it marks VALUEs in the Ruby stack, which
currently contains all of the heap memory! Somewhere in
the vast heap memory, it finds and dereferences a NULL value
and BOOM! a segfault occurs. :-)

To solve this problem, the ruby_bind_stack() function corrects

Ruby's stack to reflect the stack boundaries of the C coroutine:

(high memory address)

0x086032a0 Ruby's stack upper boundary and also
System V context's stack upper boundary

0x08601680 Ruby's stack lower boundary
(after update by SET_STACK_END)

0x082032a0 System V context's stack lower boundary

(low memory address)

Now, when the Ruby GC runs, it marks VALUEs in the correct
memory region. It does not travel into heap memory and
cause a segfault.

That is all. Thanks for reading!
=end

#19 - 11/10/2009 10:51 PM - romanbsd (Roman Shterenzon)

=begin
Hi,

I would like to say that without applying this patch, my ruby interpreter, embedded in a pthread, would cause a segmentation fault as soon as GC was invoked. I would like to see this applied to 1.9.1 as well as http://redmine.ruby-lang.org/issues/show/2279 . Without these, it's hardly possible to have ruby 1.9.1 embedded in a useful way.

Thanks,
--Roman

----- Original Message ----
From: Suraj Kurapati redmine@ruby-lang.org
To: ruby-core@ruby-lang.org
Sent: Thu, November 5, 2009 9:27:17 AM
Subject: [ruby-core:26550] [Feature #2294] [PATCH] ruby_bind_stack() to embed Ruby in coroutine

Issue #2294 has been updated by Suraj Kurapati.

Hi,

According to Matz's suggestion in [ruby-core:25139], I wrote
a detailed explanation of the problem this patch solves. I
hope this explanation is helpful. Please do not hesitate
to ask for clarifications or to correct any misunderstandings.

Thanks for your thoughtful consideration.

== Introduction

The patch adds a ruby_bind_stack() function to the Ruby C API.
This function allows the person who is embedding Ruby to
tell the Ruby GC about the stack boundaries of the embedded
environment:

void ruby_bind_stack(VALUE *lower_bound, VALUE *upper_bound);

In order to understand why this function is important, please
consider the following two modes of operation: normal & embedded.

== Normal operation: Ruby runs in a C program's main()

Initially, Ruby assumes that the stack of Ruby's main
thread exists in a high memory address range, like this:

(high memory address)

0xc1bff1f0 Ruby's stack upper boundary

0xbffff1f0 Ruby's stack lower boundary

(low memory address)

As Ruby runs, the lower boundary is adjusted (by the
SET_STACK_END macro) to reflect the machine stack pointer:

(high memory address)

0xc1bff1f0 Ruby's stack upper boundary (not changed)

0xc0ff1e80 Ruby's stack lower boundary
(after update by SET_STACK_END)

(low memory address)

== Embedded operation: Ruby runs inside a C coroutine

Initially, Ruby assumes that the stack of Ruby's main
thread exists in a high memory address range, like this:

(high memory address)

0xc1bff1f0 Ruby's stack upper boundary

0xbffff1f0 Ruby's stack lower boundary

(low memory address)

However, the stack of the C coroutine (which runs Ruby)
exists at a low memory address range, because it is
statically allocated:

(high memory address)

0xc1bff1f0 Ruby's stack upper boundary

0xbffff1f0 Ruby's stack lower boundary

0x086032a0 System V context's stack upper boundary

0x082032a0 System V context's stack lower boundary

(low memory address)

As Ruby runs, the lower boundary is adjusted (by the
SET_STACK_END macro) to reflect the machine stack pointer:

(high memory address)

0xc1bff1f0 Ruby's stack upper boundary

0x086032a0 System V context's stack upper boundary

0x08601680 Ruby's stack lower boundary
(after update by SET_STACK_END)

0x082032a0 System V context's stack lower boundary

(low memory address)

See the problem? Ruby's stack and the C coroutine stack
do not agree. They overlap!

This situation becomes worse (and causes a segfault) when
the Ruby GC runs: it marks VALUEs in the Ruby stack, which
currently contains all of the heap memory! Somewhere in
the vast heap memory, it finds and dereferences a NULL value
and BOOM! a segfault occurs. :-)

To solve this problem, the ruby_bind_stack() function corrects
Ruby's stack to reflect the stack boundaries of the C coroutine:

(high memory address)

0x086032a0 Ruby's stack upper boundary and also
System V context's stack upper boundary

0x08601680 Ruby's stack lower boundary

(after update by SET_STACK_END)

0x082032a0 System V context's stack lower boundary

(low memory address)

Now, when the Ruby GC runs, it marks VALUEs in the correct
memory region. It does not travel into heap memory and
cause a segfault.

That is all. Thanks for reading!

---

http://redmine.ruby-lang.org/issues/show/2294

---

http://redmine.ruby-lang.org

=end

#20 - 11/18/2009 03:55 PM - sunaku (Suraj Kurapati)

=begin
Hi,

Sorry to be impatient, but has there been any
further decision or consideration about this patch?

The only feedback I've received so far is that:

- An early version of this patch did not support
  multi-threading (thanks to Mr. Nobu).
- A later version of this patch worked for embedding
  Ruby 1.9 inside a pthread (thanks to Mr. Roman).

The silent suspense is "killing" me, so to speak. :-)

Thanks for your consideration.
=end

#21 - 11/18/2009 05:20 PM - nobu (Nobuyoshi Nakada)

=begin
Hi,

At Wed, 18 Nov 2009 15:55:07 +0900,
Suraj Kurapati wrote in [ruby-core:26797]:

> Sorry to be impatient, but has there been any
> further decision or consideration about this patch?

Sorry to be late.

> The only feedback I've received so far is that:
>
> - An early version of this patch did not support
>   multi-threading (thanks to Mr. Nobu).
> - A later version of this patch worked for embedding
>   Ruby 1.9 inside a pthread (thanks to Mr. Roman).

Switching stack using setcontext() can't work on all platforms.
For instance, on NetBSD and older LinuxThread stack address is
tightly bound to thread, and can't be changed. That is, your
strategy is not portable.

Why don't you simply use a thread instead?

--
Nobu Nakada

=end

#22 - 11/18/2009 06:27 PM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

- % Done changed from 0 to 100

=begin
This issue was solved with changeset r25842.
Suraj, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

=end

#23 - 11/18/2009 06:41 PM - nobu (Nobuyoshi Nakada)

- Status changed from Closed to Open

=begin
Sorry, mistaken.
=end

#24 - 11/18/2009 06:42 PM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Rejected

=begin
I don't think it's good idea to it as public API.
=end

#25 - 11/18/2009 11:15 PM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:26803] Feature #2294 [PATCH] ruby_bind_stack() to embed Ruby in coroutine"
on Wed, 18 Nov 2009 18:42:31 +0900, Nobuyoshi Nakada redmine@ruby-lang.org writes:

|I don't think it's good idea to it as public API.

Hmm, but you still should not ignore the fact described in
[ruby-core:26661]. If the patch solve a serious problem under one
condition (SEGV on embedding environment), you cannot reject it just
saying 'not a good idea'.

```
                          matz.
```

=end

#26 - 11/19/2009 04:30 AM - sunaku (Suraj Kurapati)

=begin
Hi,

Nobu Nakada wrote:

> Switching stack using setcontext() can't work on all platforms.
> For instance, on NetBSD and older LinuxThread stack address is
> tightly bound to thread, and can't be changed. That is, your
> strategy is not portable.

You are referring only to my System V context example, right? If so,
please note that I also provided a second example that uses libpcl[1]
which "can use either the ucontext.h functionalities... or the standard
longjmp()/setjmp()" and "is easily portable on almost every Unix system
and on Windows" [1].

I will create a thrid example that uses libpthread to demonstrate
how this patch lets you embed Ruby 1.9 inside a pthread. (Note that this
patch has already allowed Mr. Roman to embed Ruby 1.9 inside a pthread.)

> Why don't you simply use a thread instead?

Do you mean embedding Ruby inside a pthread?

Thanks for your consideration.

=end

#27 - 11/19/2009 04:54 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:26816] [Feature #2294] [PATCH] ruby_bind_stack() to embed Ruby in coroutine"
on Thu, 19 Nov 2009 04:30:49 +0900, Suraj Kurapati redmine@ruby-lang.org writes:

|You are referring only to my System V context example, right? If so,
|please note that I also provided a second example that uses libpcl[1]
|which "can use either the ucontext.h functionalities... or the standard
|longjmp()/setjmp()" and "is easily portable on almost every Unix system
|and on Windows" [1].

As far as I understand, libpcl is under GPL, that cannot be used in
the core Ruby. Since Ruby is not covered by GPL only.

```
                          matz.
```

=end

#28 - 11/19/2009 06:15 AM - sunaku (Suraj Kurapati)

=begin
Hi,

matz writes:

> Suraj Kurapati writes:
> |please note that I also provided a second example that uses libpcl[1]
>
> As far as I understand, libpcl is under GPL, that cannot be used in
> the core Ruby. Since Ruby is not covered by GPL only.

My patch simply adds a ruby_bind_stack() method to the Ruby C API:

ruby_bind_stack_r25604.patch (attached to Feature #2294)

It does *not* use ucontext, libpcl, pthreads or any other coroutine
libraries. These libraries are only used in the example test cases
I provided to demonstrate how the ruby_bind_stack() function can be
used to embed Ruby inside a coroutine environment:

ruby-libpcl-dynamic-stack.tgz (attached to Feature #2294)
ruby-libpcl-static-stack.tgz (attached to Feature #2294)
ruby-ucontext-static-stack.tgz (attached to Feature #2294)
ruby-ucontext-dynamic-stack.tgz (attached to Feature #2294)

So there is no problem about license compatibility, right?

Thanks for your consideration.
=end

#29 - 11/19/2009 06:51 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:26818] [Feature #2294] [PATCH] ruby_bind_stack() to embed Ruby in coroutine"
on Thu, 19 Nov 2009 06:15:50 +0900, Suraj Kurapati redmine@ruby-lang.org writes:

|> As far as I understand, libpcl is under GPL, that cannot be used in
|> the core Ruby. Since Ruby is not covered by GPL only.
|
|My patch simply adds a ruby_bind_stack() method to the Ruby C API:

|
| ruby_bind_stack_r25604.patch (attached to Feature #2294)

I am sorry about my misunderstanding. In that case, there should not be any license issue. I'd wait Nobu to express his opinion.

```
                          matz.
```

=end

#30 - 11/19/2009 05:05 PM - sunaku (Suraj Kurapati)

- File example-pthread-static.tgz added

- File example-pthread-static.tgz added

=begin
Hi,

Suraj Kurapati wrote:

> I will create a thrid example that uses libpthread to demonstrate how this patch lets you embed Ruby 1.9 inside a pthread.

As promised, I am attaching two new example test cases:

```
example-pthread-static.tgz  (statically allocated stack)
example-pthread-malloc.tgz  (dynamically allocated stack)
```

These tarballs follow the same structure as the previous examples I have attached to this issue. Nevertheless, there is also a README inside the tarballs which explains that structure.

By the way, since Ruby trunk currently has only two native threading implementations: libpthread and Win32 threads, it seems I have now demonstrated all portable ways of embedding Ruby (right?):

- System V context (directly and also through libpcl)
- setjmp/longjmp (through libpcl)
- libpthread

Will this justify the acceptance of my ruby_bind_stack() patch? :-)

Thanks for your consideration.
=end

#31 - 11/20/2009 03:46 AM - sunaku (Suraj Kurapati)

- File example-pthread-malloc.tgz added

=begin
Whoops, I attached the same static allocation example twice.

I'm attaching the dynamic allocation example this time.
=end

#32 - 11/29/2009 06:06 PM - sunaku (Suraj Kurapati)

=begin
Hi,

This issue is marked as "Rejected", but it seems that Matz agreed that this issue still needs further consideration, so please reset this issue's status to "Open".

Thanks.
=end

#33 - 11/30/2009 01:43 AM - ujihisa (Tatsuhiro Ujihisa)

- Status changed from Rejected to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

- % Done changed from 100 to 50

=begin

=end

#34 - 11/30/2009 12:33 PM - ammar (Ammar Ali)

=begin
Encountered this problem while embedding Ruby in a pthread'ed plug-in. I applied the patch against r25604 but unfortunately it did not solve the problem. When pthread_main_np() == 0 it crashes. When pthread_main_np() != 0 it works.

I will be ahppy to help test this and provide more information if needed. Thanks.

=end

#35 - 11/30/2009 01:35 PM - sunaku (Suraj Kurapati)

=begin
Hi Ammar,

That is an interesting discovery, because the patch only corrects the
stack boundaries of Ruby's main thread (when pthread_main_np() == 1).

It deliberately ignores non-main threads when doing the correction
because non-main threads are also pthreads and AFAIK they allocate
and manage their own stacks on the heap.

If possible, please modify the example-pthread-*.tgz examples to
demonstrate the failure you described and attach them to this issue.

Thanks for your consideration.
=end

#36 - 12/02/2009 07:40 AM - ammar (Ammar Ali)

=begin
Hi Suraj,

Perhaps I wishfully believed your patch to be the needed solution, but what I'm seeing is definitely occurring when pthread_main_np() == 0.

With 1.9.1-p243 I was seeing random problems that seemed stack related. After searching through the reported issues, I decided to try r25604 and that's when the error became exactly, and consistently, what you described in #2258. That's how I found this patch.

After looking at your examples I don't think they mirror my situation and would need more than a modification to make them do so. Unfortunately I will not have the time for building a test case that mimics my situation accurately before another week or two.

I'm not sure this makes any difference really, but in my case Ruby is embedded inside a dynamically loadable plug-in. The target host program comes in two editions; a client that runs its plug-ins from the main thread, and a server that runs them in a child thread. The client works fine while the server crashes on the first call to require. To create an accurate test I think I need to reproduce this exact situation. I'm not sure though, I have to investigate this when time allows.

Thank you for the hard work and your consideration.
=end

#37 - 04/02/2010 08:34 AM - znz (Kazuhiro NISHIYAMA)

- Target version changed from 1.9.2 to 2.0.0

=begin

=end

#38 - 05/25/2010 03:39 PM - sunaku (Suraj Kurapati)

=begin
Hi Ammar,

Please try r25842 or newer (with and without my patch) and see if it solves
your problem. That particular revision solves the "[BUG] object allocation
during garbage collection phase" error (reported in #2258) you encountered.

Hopefully, yours will turn out to be an unrelated issue so I can make
forward progress on getting this patch accepted (someday!!). :-)

Thanks for your consideration.
=end

#39 - 05/26/2010 04:25 AM - sunaku (Suraj Kurapati)

=begin
Hi Nobu,

I combined the various coroutine library examples into a single one:

http://github.com/sunaku/ruby-coroutine-example

You can run the example like this:

```
# libpcl
sh run.sh pcl static path_to_your_ruby_svn_trunk_installation
sh run.sh pcl dynamic path_to_your_ruby_svn_trunk_installation

# POSIX threads
sh run.sh pthread static path_to_your_ruby_svn_trunk_installation
sh run.sh pthread dynamic path_to_your_ruby_svn_trunk_installation

# System V contexts
sh run.sh ucontext static path_to_your_ruby_svn_trunk_installation
sh run.sh ucontext dynamic path_to_your_ruby_svn_trunk_installation
```

I tried this on ruby 1.9.3dev (2010-05-25 trunk 28007) [i686-linux]
with my ruby_bind_stack patch applied and observed that the patch is
still necessary to make this example work.

You can actually test multiple Ruby versions by passing:

```
sh run.sh ... ~/.multiruby/install/*
```

Please retry this example when you have a chance.

Thanks for your consideration.
=end

#40 - 05/26/2010 04:31 AM - sunaku (Suraj Kurapati)

- File ruby_bind_stack_r28007.patch added

=begin

=end

#41 - 08/14/2010 10:40 AM - sunaku (Suraj Kurapati)

- File ruby_bind_stack_r28972.patch added

=begin
Hi,

I am attaching an updated patch against SVN r28972.

Thanks for your consideration.
=end

#42 - 08/22/2010 03:26 AM - sunaku (Suraj Kurapati)

- File ruby_bind_stack_1.9.2p0.patch added

=begin
Hi,

I am attaching my patch rebased against the new Ruby 1.9.2p0 release.

Should I just continue rebasing my patch against Ruby trunk/releases
periodically like this until someone really considers this patch again?

"Slow and steady wins the race", I hope. :-)

Thanks for your consideration.

=end

#43 - 12/16/2010 12:27 AM - kouteiheika (Anonymous Anonymous)

=begin
Hi,

Could we finally get this patch commited, please? It's not like it's a thousand line behemoth and it solves a very real problem - it's impossible to embed Ruby into a pthread without it. I really see no reason not to commit this.
=end

#44 - 12/16/2010 12:37 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

Ko1, could you respond to this issue, please? Either positively or
negatively, we should not leave this untouched.

```
                          matz.
```

In message "Re: [ruby-core:33727] [Ruby 1.9-Feature#2294] [PATCH] ruby_bind_stack() to embed Ruby in coroutine"
on Thu, 16 Dec 2010 00:27:40 +0900, Anonymous Anonymous redmine@ruby-lang.org writes:

|Could we finally get this patch commited, please? It's not like it's a thousand line behemoth and it solves a very real problem - it's impossible to embed Ruby into a pthread without it. I really see no reason not to commit this.
|http://redmine.ruby-lang.org/issues/show/2294

=end

#45 - 12/16/2010 02:48 AM - ko1 (Koichi Sasada)

=begin
Hi,

Suraj, I'm sorry for late response. I missed this thread.

I read the last patch:
http://redmine.ruby-lang.org/attachments/download/1153

and I need to say "no".

As nobu said at first, this patch is not considering the multi-threading.
(and using global variables should not be accepted :) The patch is too
ad-hoc modification)

I propose another API.

idea 1:

```
// API for C extension.
// User needs to know thread value.
rb_thread_set_stack(VALUE thread_val, upper, lower) {
th = thread_data(thread_val);
th->upper = upper;
th->lower = lower;
}
```

idea 2:

```
// API called from not a Ruby world
ruby_bind_stack_for_current_native_thread(upper, lower) {
th = thread_data_for_current_native_thread();

if (th == 0) {
// Current native thread does not have
// the related ruby thread.
return 0;
}

th->upper = upper;
th->lower = lower;
return 1;
}
```

BTW, how to get the correct "upper"/"lower" address of stack?

(2010/12/16 0:37), Yukihiro Matsumoto wrote:

> Hi,
>
> Ko1, could you respond to this issue, please? Either positively or
> negatively, we should not leave this untouched.
>
> matz.
>
> In message "Re: [ruby-core:33727] [Ruby 1.9-Feature#2294] [PATCH] ruby_bind_stack() to embed Ruby in coroutine"
> on Thu, 16 Dec 2010 00:27:40 +0900, Anonymous Anonymous redmine@ruby-lang.org writes:
>
> |Could we finally get this patch commited, please? It's not like it's a thousand line behemoth and it solves a very real problem - it's impossible to embed Ruby into a pthread without it. I really see no reason not to commit this.
> |http://redmine.ruby-lang.org/issues/show/2294

--
// SASADA Koichi at atdot dot net

=end

#46 - 12/16/2010 07:46 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:33730] Re: [Ruby 1.9-Feature#2294] [PATCH] ruby_bind_stack() to embed Ruby in coroutine"
on Thu, 16 Dec 2010 02:48:07 +0900, SASADA Koichi ko1@atdot.net writes:

```
|
|Hi,
|
|Suraj, I'm sorry for late response.  I missed this thread.
|
|I read the last patch:
|http://redmine.ruby-lang.org/attachments/download/1153
|
|and I need to say "no".
|
|As nobu said at first, this patch is not considering the multi-threading.
|(and using global variables should not be accepted :)  The patch is too
|ad-hoc modification)
|
|
|I propose another API.
|
|idea 1:
|
|// API for C extension.
|// User needs to know thread value.
|rb_thread_set_stack(VALUE thread_val, upper, lower) {
|  th = thread_data(thread_val);
|  th->upper = upper;
|  th->lower = lower;
|}
|
|idea 2:
|
|// API called from not a Ruby world
|ruby_bind_stack_for_current_native_thread(upper, lower) {
|  th = thread_data_for_current_native_thread();
|
|  if (th == 0) {
|    // Current native thread does not have
|    // the related ruby thread.
|    return 0;
|  }
|
|  th->upper = upper;
|  th->lower = lower;
|  return 1;
|}
|
```

|
|BTW, how to get the correct "upper"/"lower" address of stack?

I am afraid that there's no portable and/or reliable way.
Conservative GC does similar thing, maybe you can steal boundary
values from it.

```
                          matz.
```

=end

#47 - 12/16/2010 02:23 PM - sunaku (Suraj Kurapati)

=begin
Hi,

SASADA Koichi wrote in post #968635:

> I read the last patch:
> http://redmine.ruby-lang.org/attachments/download/1153
>
> and I need to say "no".
>
> As nobu said at first, this patch is not considering the
> multi-threading.
> (and using global variables should not be accepted :) The patch is too
> ad-hoc modification)

Thanks for your feedback! I must confess that I did not really understand how my patch did not support multi-threading, but after reading your proposed API, I finally understand what Nobu was talking about. :)

I agree with your feeling and I would like to follow your proposed API.

Thanks for your consideration.

--
Posted via http://www.ruby-forum.com/.

=end

#48 - 12/16/2010 09:58 PM - ko1 (Koichi Sasada)

=begin
(2010/12/16 14:23), Suraj Kurapati wrote:

> Thanks for your feedback! I must confess that I did not really
> understand how my patch did not support multi-threading, but after
> reading your proposed API, I finally understand what Nobu was talking
> about. :)
>
> I agree with your feeling and I would like to follow your proposed API.

Could you give me a concrete example? (Execution flow)
(I'm sorry if I missed the example you already posted)

--
// SASADA Koichi at atdot dot net

=end

#49 - 12/17/2010 07:28 PM - sunaku (Suraj Kurapati)

=begin
SASADA Koichi wrote in post #968830:

> Could you give me a concrete example? (Execution flow)

Please see http://www.ruby-forum.com/topic/198119#866383

> (I'm sorry if I missed the example you already posted)

I also posted an example demonstration here:
http://www.ruby-forum.com/topic/198119#914657

Thanks for your consideration.

--
Posted via http://www.ruby-forum.com/.

=end

#50 - 11/09/2011 08:37 AM - sunaku (Suraj Kurapati)

I have updated the patch against ruby-trunk:
https://github.com/sunaku/ruby/commit/137092768af325827f3d0764325713ec51387218

It is in my branch here:
https://github.com/sunaku/ruby/tree/2294_bind_stack

#51 - 12/08/2011 10:25 AM - SteveRT (Steve Hart)

Hi
While looking for a solution to an issue we have with embedding ruby into a pthread I found this thread
We have been running ruby embedded into a pthread for about 10 years now and and have upgraded periodically. We recently moved from 1.8.7 to 1.9.3 and now experience a segv immediately rb_gc is called in mark_locations_array. The symptoms resemble those described above so we applied Suraj's patch and it solves the issue. We only have one ruby instance running and do not use ruby threads within the ruby code.

Could I respectfully ask what the status of this patch is, or what plans, if any, there are to solve this problem? Ruby has proven to be extremely powerful within our applications and we are keen to maintain currency and would prefer, for obvious reasons, not to have to patch the code.

Also - what changed between 1.8.7 and 1.9.3 to cause this issue to appear?

Thanks for your consideration and for a truly great language.
Steve

#52 - 03/16/2012 03:15 AM - sunaku (Suraj Kurapati)

Hello Steve,

I'm glad my patch fixed the problem for you. :)

Regarding what changed in Ruby 1.9 to cause this problem, see this earlier response:
https://bugs.ruby-lang.org/issues/2294#note-3

And to really understand the problem in Ruby 1.9, see this earlier explanation:
https://bugs.ruby-lang.org/issues/2294#note-18

Cheers.

#53 - 07/11/2012 03:52 AM - sunaku (Suraj Kurapati)

Hello,

This patch still works as-is with ruby-1.9.3-p194 and ruby-1.9.2-p320:

https://github.com/sunaku/ruby/compare/trunk...2294_bind_stack.patch

My coroutine example still needs this patch to succeed under Ruby 1.9:

https://github.com/sunaku/ruby-coroutine-example

Thanks for your consideration.

#54 - 10/27/2012 05:04 AM - ko1 (Koichi Sasada)

- Description updated

- Assignee changed from matz (Yukihiro Matsumoto) to ko1 (Koichi Sasada)

- Priority changed from Normal to 5

I need to read this patch carefully. Sorry for long absent.

#55 - 11/24/2012 09:28 AM - mame (Yusuke Endoh)

Ko1, what's the status?

--
Yusuke Endoh mame@tsg.ne.jp

#56 - 11/26/2012 09:03 AM - ko1 (Koichi Sasada)

I will check it *after* preview2.
This ticket will not affect *Ruby* level spec.

#57 - 01/09/2013 06:27 PM - kkaempf (Klaus Kämpf)

I can reliably reproduce the bug in my embedded Ruby project (https://github.com/kkaempf/cmpi-bindings) where Ruby (ruby-1.9.3-p194; same for git master) segfaults in mark_locations_array() accessing a memory location *within* ruby_stack_lower_bound and ruby_stack_upper_bound.

I can also verify that applying the ruby_bind_stack patch fixes the issue.

#58 - 01/23/2013 01:39 PM - ko1 (Koichi Sasada)

- Status changed from Assigned to Closed

- % Done changed from 50 to 100

This issue was solved with changeset r38905.
Suraj, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- thread_pthread.c (ruby_init_stack): ignore `STACK_END_ADDRESS' if Ruby interpreter is running on co-routine. [Feature #2294] https://bugs.ruby-lang.org/issues/2294#note-18

#59 - 02/13/2013 07:28 AM - sunaku (Suraj Kurapati)

Hello ko1,

Thanks for committing changeset r38905 into Ruby 2.0.0-rc2.

But, I am seeing the following problems with that changeset:

1. Cannot bind Ruby to a pre-allocated stack address range.

2. In my coroutine example, only ucontext works correctly:

   see https://github.com/sunaku/ruby-coroutine-example

   ```
   sh run.sh ucontext dynamic ~/.rvm/rubies/ruby-2.0.0-rc2  # OK
   sh run.sh ucontext static ~/.rvm/rubies/ruby-2.0.0-rc2   # OK
   sh run.sh libpcl dynamic ~/.rvm/rubies/ruby-2.0.0-rc2    # NG: infinite loop
   sh run.sh libpcl static ~/.rvm/rubies/ruby-2.0.0-rc2     # NG: infinite loop
   sh run.sh pthread dynamic ~/.rvm/rubies/ruby-2.0.0-rc2   # NG: infinite loop
   sh run.sh pthread static ~/.rvm/rubies/ruby-2.0.0-rc2    # NG: infinite loop
   ```

Sorry for the late feedback. (Even though I am "watching" this issue in the
Ruby issue tracker, it did not notify me by e-mail about your updates.)

Thanks for your consideration.

#60 - 02/13/2013 08:45 AM - ko1 (Koichi Sasada)

- Status changed from Closed to Assigned

> 1. Cannot bind Ruby to a pre-allocated stack address range.

I'm not sure what you are saying. ucontext' version dopre-allocation'.

> 1. In my coroutine example, only ucontext works correctly:
>
>   **see** https://github.com/sunaku/ruby-coroutine-example
>
>    ```
>    sh run.sh ucontext dynamic ~/.rvm/rubies/ruby-2.0.0-rc2  # OK
>    ```

```
sh run.sh ucontext static ~/.rvm/rubies/ruby-2.0.0-rc2   # OK
sh run.sh libpcl dynamic ~/.rvm/rubies/ruby-2.0.0-rc2    # NG: infinite loop
sh run.sh libpcl static ~/.rvm/rubies/ruby-2.0.0-rc2     # NG: infinite loop
sh run.sh pthread dynamic ~/.rvm/rubies/ruby-2.0.0-rc2   # NG: infinite loop
sh run.sh pthread static ~/.rvm/rubies/ruby-2.0.0-rc2    # NG: infinite loop
```

I checked pthread and it okay...

I'll check again soon. Please wait.

> Sorry for the late feedback. (Even though I am "watching" this issue in the
> Ruby issue tracker, it did not notify me by e-mail about your updates.)

I should notice any comments. Sorry.

#61 - 02/13/2013 02:49 PM - sunaku (Suraj Kurapati)

ko1 (Koichi Sasada) wrote:

> 1. Cannot bind Ruby to a pre-allocated stack address range.
>
> I'm not sure what you are saying.

Changeset r38905 did not include my ruby_bind_stack() function.

Without that function (or something similar), I cannot tell Ruby 2.0.0-rc2
about the stack address range of the coroutine which is hosting the Ruby.

And if Ruby is not told that information, it might make a fatal mistake:
Ruby might go beyond the coroutine's pre-allocated stack address range.

This can cause memory corruption, segfault crashes, and undefined behavior.

> ucontext' version dopre-allocation'.

Yes, in my ruby-coroutine-example application, all versions do pre-allocation.

However, Ruby must also be told (bound to) the coroutine's stack address range.

For your reference:

- libpcl coroutine is bound to pre-allocated stack here:
  https://github.com/sunaku/ruby-coroutine-example/blob/master/main.c#L204
- pthread coroutine is bound to pre-allocated stack here:
  https://github.com/sunaku/ruby-coroutine-example/blob/master/main.c#L218-L223
- ucontext coroutine is bound to pre-allocated stack here:
  https://github.com/sunaku/ruby-coroutine-example/blob/master/main.c#L235-L236
- Ruby must also be bound to coroutine's pre-allocated stack here:
  https://github.com/sunaku/ruby-coroutine-example/blob/master/main.c#L144-L150

> > 1. In my coroutine example, only ucontext works correctly:
> >
> > I checked pthread and it okay...
> >
> > I'll check again soon. Please wait.

Sure, thanks for your consideration.

#62 - 02/13/2013 03:53 PM - ko1 (Koichi Sasada)

Hi,

(2013/02/13 14:49), sunaku (Suraj Kurapati) wrote:

> ko1 (Koichi Sasada) wrote:
>
> > > 1. Cannot bind Ruby to a pre-allocated stack address range.
> >
> > I'm not sure what you are saying.
>
> Changeset r38905 did not include my ruby_bind_stack() function.
>
> Without that function (or something similar), I cannot tell Ruby 2.0.0-rc2
> about the stack address range of the coroutine which is hosting the Ruby.
>
> And if Ruby is not told that information, it might make a fatal mistake:
> Ruby might go beyond the coroutine's pre-allocated stack address range.
>
> This can cause memory corruption, segfault crashes, and undefined behavior.
>
> > ucontext' version dopre-allocation'.
>
> Yes, in my ruby-coroutine-example application, all versions do pre-allocation.
>
> However, Ruby must also be told (bound to) the coroutine's stack address range.

I think you got misunderstanding.

(1) Ruby interpreter must know machine stack start, not a `range'.
(2) Machine stack start can be grabbed without telling an address
range explicitly.
See a r38905.

NOTE: On Ruby 1.8, may run fine on co-routines.
The patch revert to this behavior.

Without machine-stack-end, we can't detect machine stack overflow. This
is an issue. But not a critical issue.

However, the implantation of `ruby_bind_stack()' has several problems:

- Designed on misunderstanding
- Depend on specific architecture

I agree to add another interface to tell the stack range. But
`ruby_bind_stack()' is not acceptable now. At least Ruby 2.0.0 shouldn't
include it.

Maybe it will be a parameter of an initialize function of the Ruby
interpreter.

> Ruby might go beyond the coroutine's pre-allocated stack address range.

Yes. That's right.

> > > 1. In my coroutine example, only ucontext works correctly:
> >
> > I checked pthread and it okay...
> >
> > I'll check again soon. Please wait.
>
> Sure, thanks for your consideration.

I believe last patch will solve this issue.
So the problem is based on my misunderstanding or a bug.
I want to solve this issue before 2.0.0 release.

I'll check it soon.

--
// SASADA Koichi at atdot dot net

#63 - 02/13/2013 11:28 PM - sunaku (Suraj Kurapati)

Hi,

ko1 (Koichi Sasada) wrote:

> (2013/02/13 14:49), sunaku (Suraj Kurapati) wrote:
>
> > Ruby must also be bound to the coroutine's stack address range.
>
> I think you got misunderstanding.
>
> (1) Ruby interpreter must know machine stack start, not a `range'.
> (2) Machine stack start can be grabbed without telling an address
> range explicitly.
> See a r38905.
>
> NOTE: On Ruby 1.8, may run fine on co-routines.
> The patch revert to this behavior.
>
> Without machine-stack-end, we can't detect machine stack
> overflow. This is an issue. But not a critical issue.
>
> However, the implantation of `ruby_bind_stack()' has several
> problems:
>
> - Designed on misunderstanding
> - Depend on specific architecture

Ah, I finally understand now. Thank you for explaining! :-)

> I agree to add another interface to tell the stack range. But
> `ruby_bind_stack()' is not acceptable now. At least Ruby 2.0.0
> shouldn't include it.
>
> Maybe it will be a parameter of an initialize function of the
> Ruby interpreter.

Very well; that is reasonable. I agree with your decision.

> > Ruby might go beyond the coroutine's pre-allocated stack
> > address range.
>
> Yes. That's right.

I'm glad to have the same understanding in this aspect.

> > 1. In my coroutine example, only ucontext works correctly:
>
> I believe last patch will solve this issue.
> So the problem is based on my misunderstanding or a bug.
> I want to solve this issue before 2.0.0 release.
>
> I'll check it soon.

Certainly; I will wait for your response.

Please tell me if I can be of assistance.

#64 - 02/24/2013 09:16 PM - ko1 (Koichi Sasada)

- Target version changed from 2.0.0 to 2.1.0

Sorry for my late work.
I'll check soon.

#65 - 01/30/2014 06:16 AM - hsbt (Hiroshi SHIBATA)

- Target version changed from 2.1.0 to 2.2.0

#66 - 01/05/2018 09:00 PM - naruse (Yui NARUSE)

- Target version deleted (2.2.0)

Files

ruby_bind_stack.patch	2.66 KB	10/28/2009	sunaku (Suraj Kurapati)
ruby-ucontext-static-stack.tgz	10 KB	10/28/2009	sunaku (Suraj Kurapati)
ruby-ucontext-dynamic-stack.tgz	10.2 KB	10/28/2009	sunaku (Suraj Kurapati)
ruby-libpcl-static-stack.tgz	10.3 KB	10/28/2009	sunaku (Suraj Kurapati)
ruby-libpcl-dynamic-stack.tgz	10 KB	10/28/2009	sunaku (Suraj Kurapati)
ruby_bind_stack.patch	2.67 KB	10/28/2009	sunaku (Suraj Kurapati)
ruby_bind_stack.patch	5 KB	10/28/2009	sunaku (Suraj Kurapati)
ruby-ucontext-thread.tgz	54.2 KB	10/28/2009	sunaku (Suraj Kurapati)
ruby-libpcl-thread.tgz	53.3 KB	10/28/2009	sunaku (Suraj Kurapati)
ruby_bind_stack.patch	4.98 KB	10/28/2009	sunaku (Suraj Kurapati)
ruby_bind_stack.patch	5.47 KB	10/30/2009	sunaku (Suraj Kurapati)
ruby_bind_stack.patch	5.45 KB	10/30/2009	sunaku (Suraj Kurapati)
ruby_bind_stack.patch	5.46 KB	10/30/2009	sunaku (Suraj Kurapati)
ruby_bind_stack.patch	5.4 KB	10/30/2009	sunaku (Suraj Kurapati)
ruby_bind_stack_after_refactoring.patch	3.06 KB	10/31/2009	sunaku (Suraj Kurapati)
ruby_bind_stack.patch	5.72 KB	11/01/2009	sunaku (Suraj Kurapati)
ruby_bind_stack_after_refactoring.patch	3.37 KB	11/01/2009	sunaku (Suraj Kurapati)
ruby_bind_stack.patch	5.76 KB	11/01/2009	sunaku (Suraj Kurapati)
ruby_bind_stack_after_refactoring.patch	3.42 KB	11/01/2009	sunaku (Suraj Kurapati)
ruby_bind_stack_r25604.patch	3.78 KB	11/04/2009	sunaku (Suraj Kurapati)
example-pthread-static.tgz	55 KB	11/19/2009	sunaku (Suraj Kurapati)
example-pthread-static.tgz	55 KB	11/19/2009	sunaku (Suraj Kurapati)
example-pthread-malloc.tgz	55.5 KB	11/20/2009	sunaku (Suraj Kurapati)
ruby_bind_stack_r28007.patch	3.78 KB	05/26/2010	sunaku (Suraj Kurapati)
ruby_bind_stack_r28972.patch	3.78 KB	08/14/2010	sunaku (Suraj Kurapati)
ruby_bind_stack_1.9.2p0.patch	3.78 KB	08/22/2010	sunaku (Suraj Kurapati)