Ruby master - Feature #9018

Make statically linked extensions easier to use

10/13/2013 05:38 AM - Ahti (Lukas S)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

Sorry for the lengthy post, my english is not that good and the idea is not that simple to explain either.

Current situation

For platforms where dynamic linking is not available, the ruby build system provides a way to disable dynamic loading of '.bundle's. When this is disabled, a program embedding libruby-static needs to link against the static library build for an extension and manually call its Init_someextension() function to use the extension.

This has some issues:

1. A program embedding libruby-static needs to link a whole bunch of static libraries.
2. The program then needs to call the corresponding Init_something() function for each extension
3. require wouldn't work as usual for the extension parts. E.g.: require 'etc' won't work, because there is no etc file in the include path. Instead the things provided by the extension will be available as soon as the embedding program calls the init function of the extension.

Improvements

I think this could be made a lot easier:

1. While building libruby-static, also link all extensions into libruby-static.a, so programs linking this library also get all extensions.
2. Add dummy files into the ruby library that tell the require function to call the Init function for the ext being required, so the extension won't be available before it is required, and can be required just like when using dynamic linking.

Details on 2.:

The require mechanism implemented in rb_require_safe (load.c) currently differentiates between two file types: a ruby file and a shared object.

I would suggest adding a third type which is just a dummy file (recognized via file extension, e.g. 'staticext').

When a file of this type is required, the require mechanism determines the corresponding Init_someextension function (either via a table that is filled at compile time,
or (and preferably) via building a function name from the filename (or contents of the file???) and getting the functions memory address
via something like dlsym).

It then calls the function if it has not already been called (it'd need to keep a list of files that were already included for this). If the function is not defined, a LoadError is raised.

The dummy files would need to be put into the ruby lib, just like bundles are put there when linking dynamically. It may be a good idea to put them in a special folder to make separation easier though.
With the example of the 'etc'-extension, such a dummy file could have a path something like this:
/usr/lib/ruby/2.0.0/linked-ext/etc.staticext.

If this was implemented, using statically linked extensions would be just like using dynamically loaded bundles from the ruby side, which is (imho) the biggest priority.

Example

To make my idea a little bit more clear, here is a example of how this might work for the extension 'etc':

Build process:

1. While compiling, the user passes the options '--disable-dln', '--with-static-linked-ext' and '--enable-shared=no' to configure, causing dynamic linking to be disallowed.
2. The user uncomments 'option nodynamic' and 'etc' in ext/Setup.
3. While building, the build system links the static library generated for etc into libruby-static.
4. The build system also places a file called 'etc.staticext' in the ruby library folder.

At runtime:

1. A script calls require 'etc'.
2. rb_require_safe finds the etc.staticext file.
3. rb_require_safe determines the init function name as Init_etc and calls this function.
4. The etc extension is usable in the script.

Associated revisions

Revision 18d8ba25 - 08/21/2015 10:48 AM - nobu (Nobuyoshi Nakada)

Makefile.sub: link libraries for extensions

- win32/Makefile.sub ($(LIBRUBY_SO)): needs additional libraries for extension libraries to link statically. [ruby-core:70499] [Feature #9018]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@51653 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 51653 - 08/21/2015 10:48 AM - nobu (Nobuyoshi Nakada)

Makefile.sub: link libraries for extensions

- win32/Makefile.sub ($(LIBRUBY_SO)): needs additional libraries for extension libraries to link statically. [ruby-core:70499] [Feature #9018]

Revision 51653 - 08/21/2015 10:48 AM - nobu (Nobuyoshi Nakada)

Makefile.sub: link libraries for extensions

- win32/Makefile.sub ($(LIBRUBY_SO)): needs additional libraries for extension libraries to link statically. [ruby-core:70499] [Feature #9018]

Revision 51653 - 08/21/2015 10:48 AM - nobu (Nobuyoshi Nakada)

Makefile.sub: link libraries for extensions

- win32/Makefile.sub ($(LIBRUBY_SO)): needs additional libraries for extension libraries to link statically. [ruby-core:70499] [Feature #9018]

Revision 51653 - 08/21/2015 10:48 AM - nobu (Nobuyoshi Nakada)

Makefile.sub: link libraries for extensions

- win32/Makefile.sub ($(LIBRUBY_SO)): needs additional libraries for extension libraries to link statically. [ruby-core:70499] [Feature #9018]

Revision 51653 - 08/21/2015 10:48 AM - nobu (Nobuyoshi Nakada)

Makefile.sub: link libraries for extensions

- win32/Makefile.sub ($(LIBRUBY_SO)): needs additional libraries for extension libraries to link statically. [ruby-core:70499] [Feature #9018]

History

#1 - 10/14/2013 12:49 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Feedback

1. A program embedding libruby-static needs to link a whole bunch of static libraries.

All static libraries are linked into libruby-static.a.

1. The program then needs to call the corresponding Init_something() function for each extension

Those functions are called at bootstrap.

I don't think a dummy file is needed.

#2 - 10/14/2013 09:23 AM - Ahti (Lukas S)

> All static libraries are linked into libruby-static.a.

Okay then I need to fix my build process. Can You tell me what flag exactly is making this happen?

> Those functions are called at bootstrap.

bootstrap = ruby_init?

> I don't think a dummy file is needed.

I think it would be nice for the following reason:

Imagine a program embeds ruby to provide some scripting functionality to the user.

With how it is now, the user of that program needs to know wether an extension is a native extension (so he won't need to and in fact can't require it) or it is just a ruby file in the lib (in which case he needs to require it normally). I think this is very inconvenient for the user and makes using the standard library more complicated and involved.

With the dummy files the behaviour of statically linked and dynamically loaded extensions would be absolutely the same to the user, so the behaviour of ruby would be consistent everywhere, which is, imho, a very important thing.

#3 - 08/21/2015 05:03 AM - scorpion007 (Alex Budovski)

I agree with Lukas.

Nobu, only the builtin modules are linked into the binary and called at bootstrap:

inits.c:

```
void
rb_call_inits(void)
{
    CALL(Method);
    CALL(RandomSeed);
    CALL(sym);
    CALL(var_tables);
    CALL(Object);
    CALL(top_self);
...
```

Not the standard library modules, like bigdecimal.so. That is linked as a separate DLL and depends on the ruby DLL.

It makes the static lib approach only half baked: it works for the builtin functionality (array, bignum, etc) but not for the standard lib.

#4 - 08/21/2015 05:13 AM - scorpion007 (Alex Budovski)

It looks like for the latter point (dummy file) we can actually use rb_provide! Ruby uses it to "fake" a module for back compat; for modules that were once external, but are now builtin. Like Complex, Rational, Enumerator.

```
void
Init_Complex(void)
{
...
    rb_provide("complex.so");   /* for backward compatibility */
```

That will trick require into thinking it found a module when it's really builtin, from what I understand.

I've yet to try it though.

#5 - 08/21/2015 05:56 AM - scorpion007 (Alex Budovski)

So actually, it looks like there's a switch --with-static-linked-ext that can be used to link all the exts statically! The win32\Makefile.sub has a bug where it's not specifying all the libs needed by win32ole ext though.

I had to add ole32.lib oleaut32.lib:

```
!if !defined(LIBS)
LIBS = user32.lib advapi32.lib shell32.lib ws2_32.lib ole32.lib oleaut32.lib
```

But this looks promising!

#6 - 08/21/2015 10:48 AM - nobu (Nobuyoshi Nakada)

- Status changed from Feedback to Closed

Applied in changeset r51653.

Makefile.sub: link libraries for extensions

- win32/Makefile.sub ($(LIBRUBY_SO)): needs additional libraries for extension libraries to link statically. [ruby-core:70499] [Feature #9018]

#7 - 08/24/2015 07:45 AM - nobu (Nobuyoshi Nakada)

- Description updated