**Proposal: Internal GC/memory subsystem API**

07/08/2011 05:50 AM - kstephens (Kurt Stephens)

<table>
<thead>
<tr>
<th>Status</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee</td>
<td>authorNari (Narihiro Nakamura)</td>
</tr>
<tr>
<td>Target version</td>
<td>2.6</td>
</tr>
</tbody>
</table>

**Description**

There is significant interest in improving/altering the performance, behavior and features of MRI's GC in 1.8 and 1.9 series.

Proposal: MRI should support an internal GC API -- to separate MRI core from its current GC implementation, and provide hooks for additional features:

1) Interfaces between MRI internals and any GC/allocator implementation:
   - stock MRI GC
   - malloc() without free() to support valgrind testing (or short-lived programs)
   - other collectors ([https://github.com/kstephens/smal](https://github.com/kstephens/smal))

2) Support selecting GC implementations at run-time or compile time.

3) Support malloc() replacements, at run-time and/or compile time, such as:
   - tcmalloc
   - jemalloc

4) Support callback hooks in allocation and GC phases to orthogonally add features, such as:
   - performant/correct WeakReferences and ReferenceQueues ([http://redmine.ruby-lang.org/issues/4168](http://redmine.ruby-lang.org/issues/4168)).
   - allocation tracing/debugging.
   - instance caching (e.g.: Floats)
   - computational caching.
   - cache invalidation.
   - metrics collection.

5) Interfaces to common features of alternate GCs:
   - finalization
   - weak references
   - atomic allocations (e.g.: string or binary data)
   - mostly read-only/static allocations (e.g.: code, global bindings)

A prototype GC phase callback API for 1.8, REE and 1.9 is here:


This GC API should be supported on both 1.8 and 1.9 code lines.

**Related issues:**

Related to Ruby master - Feature #2471: want to choose a GC algorithm  
Rejected 12/10/2009

**History**

#1 - 07/08/2011 06:55 AM - matz (Yukihiro Matsumoto)

This GC API should be supported on both 1.8 and 1.9 code lines.

There's no chance to add new API to 1.8. 1.8 has been dead new-feature-wise-ly.
I've made a small amount of progress on a prototype. I will post links to github branches ASAP.

Branch is here: [https://github.com/kstephens/ruby/tree/trunk-mem-api](https://github.com/kstephens/ruby/tree/trunk-mem-api)

Supports boot-time selection between the standard gc.c memory system (named "core") and a malloc-only system (named "malloc"); the "core" memory system is default.

Build:

```
./configure --prefix=... && make
```

Then try:

```
RUBY_MEM_SYS=malloc make install
```

or

```
RUBY_MEM_SYS=malloc:D make install # produces debug output using malloc-only.
RUBY_MEM_SYS=malloc:DL make install # produces debug output using malloc-only with @file:line.
```

Features not yet implemented: hooks for finalization, add hooks in gc.c for GC callbacks (weak reference/reference queue support).

Why don't you use GC.disable?

Because GC.disable doesn't leverage memory debuggers (valgrind) effectively; gc.c still continues to allocate large chunks (heaps) via malloc(); thus valgrind can not distinguish between (and instrument) RVALUEs that are parceled from the large chunks in rb_newobj().

Also because this is a proof-of-concept prototype.


I'm not "blaming" valgrind, I'm not even "blaming" gc.c. Valgrind has hooks for recognizing custom allocators, but only if the allocator is instrumented. It's obvious, MRI 1.9 uses some of the VALGRIND~*() hooks, however the hooks are enabled at compile-time.
Again, this is part of a proof-of-concept -- that different allocators and features can be selected at boot-time (and/or compile-time). This is not the final work -- there is a larger goal stated in the description.

BTW: using the RUBY_MEM_SYS=malloc allocator during “make” and “make install” appears to improve speed with only a modest memory increase. Some ruby programs are short-lived and do not create much collectable garbage.

If anyone thinks the larger goal, or even this dinky "malloc-only" allocator, is useful, please speak up. :)