Proposal: Improved Finalizer Semantics

ObjectSpace.define_finalizer(object, proc):

** proc should have a single parameter, the object to be finalized, not its id.

When an object with a finalizer is no longer referenced (sweepable):

- The object is reconsidered to be REFERENCED until next GC.
- It's finalizer proc(s) are called, only once, with the object as the sole argument.
- Subsequently, the finalizer procs are removed from the object.
- The object's memory will NOT be reclaimed yet, nor will its C free_proc be called, since calling the finalizer proc effectively creates new (temporary) references to the object.
- If the finalizer did NOT create any additional, long-term references to the object, the object's memory and low-level C resources will be reclaimed in the next GC.

This is a simpler protocol:

- It removes the need for _id2ref in the finalizer procs.
- Prevents other complications: such as GC being reinvoked within a finalizer.
- Finalizers are invoked with the same "urgency" as before.

The downside:

- Objects with finalizers actually live for more than one GC cycle, if they are unreferenced.
- This is probably acceptable since the resources the finalizers "clean-up" (eg.: file descriptors in a File object) are generally more scarce than the objects holding them.

History

#1 - 10/28/2011 12:53 PM - matz (Yukihiro Matsumoto)

Hi,

In message “Re: [ruby-core:40474] [ruby-trunk - Feature #5494] [Open] Proposal: Improved Finalizer Semantics” on Fri, 28 Oct 2011 12:20:38 +0900, Kurt Stephens ks.ruby@kurtstephens.com writes:

[This is a simpler protocol:
| |
| " It removes the need for _id2ref in the finalizer procs.
| " Prevents other complications: such as GC being reinvoked within a finalizer.
| " Finalizers are invoked with the same "urgency" as before.

- _id2ref does not work in finalizers, since their corresponding objects are already freed.
- finalizers are called after GC.

Your proposal might be based on false assumption. If I were going to make an incompatible change to the finalizer in the future, I'd remove it altogether.

matz.

#2 - 10/28/2011 02:16 PM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Rejected

Even if the object has not been freed, other objects referred by the object might be freed already. It has to sort topologically the
directed graph, even for simple cases. Not only this would be quite expensive, cyclic references cannot sort.

#3 - 10/28/2011 02:35 PM - kstephens (Kurt Stephens)
Nobuyoshi Nakada wrote:

Even if the object has not been freed, other objects referred by the object might be freed already. It has to sort topologically the directed graph, even for simple cases. Not only this would be quite expensive, cyclic references cannot sort.

Good point. It would have to re-MARK the object and its references, recursively, to be safe.

#4 - 11/02/2011 03:57 AM - kstephens (Kurt Stephens)
Kurt Stephens wrote:

Nobuyoshi Nakada wrote:

Even if the object has not been freed, other objects referred by the object might be freed already. It has to sort topologically the directed graph, even for simple cases. Not only this would be quite expensive, cyclic references cannot sort.

Good point. It would have to re-MARK the object and its references, recursively, to be safe.

Clarifications:

- The idea is to run finalizers (and re-MARK) after root marking, and before sweep. Thus, it would work with lazy sweep.
- The MARK operation on the finalized object is the exact same recursive MARK operation that already exists in gc.c.
- The semantics I described is exactly how (most) JVMs implement finalizations. It's also how I implemented finalizations in SMAL.
- We can keep the proc.call(obj.id) API, until the proc.call(obj) API change is acceptable.