Ruby master - Feature #13512

System Threads

04/26/2017 02:16 PM - magaudet (Matthew Gaudet)

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

Description

In this feature I propose creating a VM implementation feature called (for lack of a better name) *System Threads*.

Background

I am working on adding asynchronous compilation to Ruby+OMR allow compilation of methods to occur off the application thread, to avoid latency jumps when a method is selected for compilation. In the process of compiling Ruby, I really need to be able to call into the interpreter, to query for information about methods, and eventually I may even want to run small pieces of Ruby code (to fold constantes for example). I cannot call into the interpeter in any way without using a ruby level thread (required to acquire the GVL), and so, today, my implementation uses the existing Ruby thread API (rb_thread_create, rb_thread_call_without_gvl2, rb_thread_call_with_gvl). This approach largely works, and I am able to boot Discourse with the JIT + Async compilation enabled, thanks to ko1.

Unfortunately, I'm unable to pass make test, a curious situation.

The issue is that because the compilation is a persistent Thread, there is some functionality that stops working (though, not enough to cause Discourse to fail). For example, the deadlock detection code tested by the following case no longer fires, and so a hang occurs:

```
assert_equal 'ok', %q{
  begin
    Thread.new { Thread.stop }
    Thread.stop
    :ng
  rescue Exception
    :ok
  end
}
```

This seems undesirable: The deadlock detection API is very useful, and I really don't want to neuter it by introducing a compilation thread. Hence the proposal for a System Thread.

System Threads

I am proposing resolving the problem above by adding a new thread API: VALUE rb_system_thread_create(VALUE (*)(ANYARGS), void*);. Later extensions could provide a subclass of Thread called SystemThread however for now I consider that mildly out of scope.

Here are the distinguishing features as I see them:

- A Thread created through the System Thread API would not show up in Thread::list (though, if SystemThread is created, would show up in its list).
- For the purposes of error reporting, System Threads would be resumed not to be partaking in user code. ie:
  - System threads aren't eligible to wake condition variables
  - System threads aren't enough to have the system able to make forward progresss.

Potential Users:

In addition to JIT compilation threads, I see potential for other future consumers:

- Concurrent garbage collection would likely need to take place on a System Thread to allow finalizers to run.
- (Potential for misuse) one could imagine some libraries would want system threads.

Other notes:

I took a little bit of time to try to implement System Threads, enough to realize that this wouldn't be a trivial implementation. There are a couple of different approaches I think that could be successful, but I didn't get deep enough into the thread code to have a strong intuition, and so I wanted to get feedback before going too far down this road.

As a side note, I also attempted an alternative to System Threads for the purposes of the compilation thread: I used a native thread, then intended to spawn a Ruby thread only at the point where the JIT had a query:

```
# This was implemented using the C-API.
def call_into_vm(query, arg)
  Thread.new { query(arg) }.value
end
```

This was unsuccessful because creating a thread requires holding the GVL, but a native thread cannot acquire the GVL as near as I can tell.

History

#1 - 04/26/2017 03:10 PM - ko1 (Koichi Sasada)

Sorry I can't understand your proposal. What is the problem and why System thread solve it?
As I said before, I'm not sure why you need to cooperate with Ruby threads.
Such compilation threads should run on native threads which are independent from Ruby threads.

For example, some years ago I wrote parallel sweep. Which invokes a native sweeping thread and it is completely independent from Ruby's thread system.

#2 - 04/26/2017 03:30 PM - magaudet (Matthew Gaudet)

So, I'd love to use a native thread to do compilation, however, that means compilation is cut off entirely from the interpreter: Any and all rb_ family functions that are only callable under the GVL become out of bounds. In the short term, this can be accomplished by changing the way we enqueue a method for compilaion, requiring that any and all VM knowledge the compiler may want be baked into the compilation request.

Longer term however, I hope you can imagine how when compiling a Ruby method you might want to call a variety of rb_ functions to better inform the JIT compiler about what it's compiling.

The reason for having 'System threads' is that we need to call into the Ruby VM, executing code under the GVL, but we don't want to disrupt the user's perspective of threaded code: If the user's code has deadlocked, it should report deadlock, if the user has no more running threads, it should error, etc.

#3 - 07/14/2017 07:08 AM - ko1 (Koichi Sasada)

- Assignee set to ko1 (Koichi Sasada)