Ruby trunk - Feature #10600

[PATCH] Queue#close

12/15/2014 09:12 AM - djellemah (John Anderson)

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

Description

In a multiple-producer / multiple-consumer situation using blocking enq and deq, closing a queue cleanly is difficult. It's possible using a queue poison token, but unpleasant because either producers have to know how to match up number of poison tokens with number of consumers, or consumers have to keep putting the poison back into the queue which complicates testing for empty and not blocking on deq.

This patch (from trunk at b2a128f) implements Queue#close which will close the queue to producers, leaving consumers to deq the remaining items. Once the queue is both closed and empty, consumers will not block. When an empty queue is closed, all consumers blocking on deq will be woken up and given nil.

With Queue#close, clean queue shutdown is simple:

```
queue = SizedQueue.new 1000

consumer_threads = lots_of.times.map do
  Thread.new do
    while item = queue.pop
      do_work item
    end
  end
end

source = somewhat_async_enumerator

producer_threads = a_few.times.map do
  Thread.new do
    loop{queue << source.next}
  end
end

producer_threads.each &:join
queue.close
consumer_threads.each &:join
```

Associated revisions

Revision fd7ac9f3 - 08/26/2015 10:59 PM - ko1 (Koichi Sasada)

- thread_tools.c: add Queue#close(exception=false) and SizedQueue#close(exception=false). [Feature #10600] Trying to deq from a closed empty queue return nil if exception parameter equals to false (default). If exception parameter is truthy, it raises ClosedQueueError (< StopIteration). ClosedQueueError inherits StopIteration so that you can write: loop{ e = q.deq; (using e) } Trying to close a closed queue raises ClosedQueueError. Blocking threads to wait deq for Queue and SizedQueue will be restarted immediately by returning nil (exception=false) or raising a ClosedQueueError (exception=true). Blocking threads to wait enq for SizedQueue will be restarted by raising a ClosedQueueError immediately. The above specification is not proposed specification, so that we need to continue discussion to conclude specification this method.
- test/thread/test_queue.rb: add tests originally written by John Anderson and modify detailed behavior.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@51699 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 51699 - 08/26/2015 10:59 PM - ko1 (Koichi Sasada)

- thread_tools.c: add Queue#close(exception=false) and SizedQueue#close(exception=false). [Feature #10600] Trying to deq from a closed empty queue return nil if exception parameter equals to false (default). If exception parameter is truthy, it raises ClosedQueueError (< StopIteration). ClosedQueueError inherits StopIteration so that you can write: loop{ e = q.deq; (using e) } Trying to close a closed queue raises ClosedQueueError. Blocking threads to wait deq for Queue and SizedQueue will be restarted immediately by returning nil (exception=false) or raising a ClosedQueueError (exception=true). Blocking threads to wait enq for SizedQueue will be restarted by raising a ClosedQueueError

immediately. The above specification is not proposed specification, so that we need to continue discussion to conclude specification this method.
- test/thread/test_queue.rb: add tests originally written by John Anderson and modify detailed behavior.

Revision 51699 - 08/26/2015 10:59 PM - ko1 (Koichi Sasada)

- thread_tools.c: add Queue#close(exception=false) and SizedQueue#close(exception=false). [Feature #10600] Trying to deq from a closed empty queue return nil if exception parameter equals to false (default). If exception parameter is truthy, it raises ClosedQueueError (< StopIteration). ClosedQueueError inherits StopIteration so that you can write: loop{ e = q.deq; (using e) } Trying to close a closed queue raises ClosedQueueError. Blocking threads to wait deq for Queue and SizedQueue will be restarted immediately by returning nil (exception=false) or raising a ClosedQueueError (exception=true). Blocking threads to wait enq for SizedQueue will be restarted by raising a ClosedQueueError immediately. The above specification is not proposed specification, so that we need to continue discussion to conclude specification this method.
- test/thread/test_queue.rb: add tests originally written by John Anderson and modify detailed behavior.

Revision 51699 - 08/26/2015 10:59 PM - ko1 (Koichi Sasada)

- thread_tools.c: add Queue#close(exception=false) and SizedQueue#close(exception=false). [Feature #10600] Trying to deq from a closed empty queue return nil if exception parameter equals to false (default). If exception parameter is truthy, it raises ClosedQueueError (< StopIteration). ClosedQueueError inherits StopIteration so that you can write: loop{ e = q.deq; (using e) } Trying to close a closed queue raises ClosedQueueError. Blocking threads to wait deq for Queue and SizedQueue will be restarted immediately by returning nil (exception=false) or raising a ClosedQueueError (exception=true). Blocking threads to wait enq for SizedQueue will be restarted by raising a ClosedQueueError immediately. The above specification is not proposed specification, so that we need to continue discussion to conclude specification this method.
- test/thread/test_queue.rb: add tests originally written by John Anderson and modify detailed behavior.

Revision 51699 - 08/26/2015 10:59 PM - ko1 (Koichi Sasada)

- thread_tools.c: add Queue#close(exception=false) and SizedQueue#close(exception=false). [Feature #10600] Trying to deq from a closed empty queue return nil if exception parameter equals to false (default). If exception parameter is truthy, it raises ClosedQueueError (< StopIteration). ClosedQueueError inherits StopIteration so that you can write: loop{ e = q.deq; (using e) } Trying to close a closed queue raises ClosedQueueError. Blocking threads to wait deq for Queue and SizedQueue will be restarted immediately by returning nil (exception=false) or raising a ClosedQueueError (exception=true). Blocking threads to wait enq for SizedQueue will be restarted by raising a ClosedQueueError immediately. The above specification is not proposed specification, so that we need to continue discussion to conclude specification this method.
- test/thread/test_queue.rb: add tests originally written by John Anderson and modify detailed behavior.

Revision 51699 - 08/26/2015 10:59 PM - ko1 (Koichi Sasada)

- thread_tools.c: add Queue#close(exception=false) and SizedQueue#close(exception=false). [Feature #10600] Trying to deq from a closed empty queue return nil if exception parameter equals to false (default). If exception parameter is truthy, it raises ClosedQueueError (< StopIteration). ClosedQueueError inherits StopIteration so that you can write: loop{ e = q.deq; (using e) } Trying to close a closed queue raises ClosedQueueError. Blocking threads to wait deq for Queue and SizedQueue will be restarted immediately by returning nil (exception=false) or raising a ClosedQueueError (exception=true). Blocking threads to wait enq for SizedQueue will be restarted by raising a ClosedQueueError immediately. The above specification is not proposed specification, so that we need to continue discussion to conclude specification this method.
- test/thread/test_queue.rb: add tests originally written by John Anderson and modify detailed behavior.

Revision e2609033 - 11/21/2015 12:32 AM - ko1 (Koichi Sasada)

- thread_sync.c: reduce the specification of Queue#close.
  - Queue#close accepts no arguments.
  - deq'ing on closed queue returns nil, always. [Feature #10600]

- test/thread/test_queue.rb: catch up this fix.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52691 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 52691 - 11/21/2015 12:32 AM - ko1 (Koichi Sasada)

- thread_sync.c: reduce the specification of Queue#close.
  - Queue#close accepts no arguments.
  - deq'ing on closed queue returns nil, always. [Feature #10600]

- test/thread/test_queue.rb: catch up this fix.

Revision 52691 - 11/21/2015 12:32 AM - ko1 (Koichi Sasada)

- thread_sync.c: reduce the specification of Queue#close.
  - Queue#close accepts no arguments.
  - deq'ing on closed queue returns nil, always. [Feature #10600]
- test/thread/test_queue.rb: catch up this fix.

Revision 52691 - 11/21/2015 12:32 AM - ko1 (Koichi Sasada)

- thread_sync.c: reduce the specification of Queue#close.
  - Queue#close accepts no arguments.
  - deq'ing on closed queue returns nil, always. [Feature #10600]
- test/thread/test_queue.rb: catch up this fix.

Revision 52691 - 11/21/2015 12:32 AM - ko1 (Koichi Sasada)

- thread_sync.c: reduce the specification of Queue#close.
  - Queue#close accepts no arguments.
  - deq'ing on closed queue returns nil, always. [Feature #10600]
- test/thread/test_queue.rb: catch up this fix.

Revision 52691 - 11/21/2015 12:32 AM - ko1 (Koichi Sasada)

- thread_sync.c: reduce the specification of Queue#close.
  - Queue#close accepts no arguments.
  - deq'ing on closed queue returns nil, always. [Feature #10600]
- test/thread/test_queue.rb: catch up this fix.

History

#1 - 12/15/2014 09:35 AM - ko1 (Koichi Sasada)

Interesting. I understand your motivation.

I have several questions (design choise)

(1) should we flush all remaining items in queue when it is closing?

This specification can be interrupt.

Now, your proposal does not flush.

(2) should we allow "re-open"?

We can make it. But it makes thread programming difficult to control.

Maybe we need to survey other language / libraries.

#2 - 12/15/2014 01:01 PM - nobu (Nobuyoshi Nakada)

(3) Shouldn't Queue#pop also raise an exception if the queue is empty and closed, instead of returning nil?

(4) What happens on another thread which is blocked at SizedQueue#push?

#3 - 12/15/2014 02:37 PM - djellemah (John Anderson)

Koichi Sasada wrote:

> Interesting. I understand your motivation.

It's always nice to be understood ;-)

> I have several questions (design choise)
>
> (1) should we flush all remaining items in queue when it is closing?
>
> This specification can be interrupt.

I'm not sure what you mean here?

> Now, your proposal does not flush.

No, because in some cases there will still be items in the queue which the consumers have not finished processing. Flush on close would mean those items would be lost.

queue.close.clear would achieve flush, but it would not be atomic.

> (2) should we allow "re-open"?
>
> We can make it. But it makes thread programming difficult to control.

I'm leaning towards no. If the queue could be re-opened, the consumer side would not know with certainty when to let consumer threads end. So the shutdown simplicity would be gone.

> Maybe we need to survey other language / libraries.

No help from the wikipedia entry - it just assumes that the producer and consumer will run forever (while (true) ...):
http://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem

Java's BlockingQueue is the same: https://docs.oracle.com/javase/6/docs/api/java/util/concurrent/BlockingQueue.html There are many stackoverflow questions on how to know when a queue is finished. Most of the answers suggest the poison pill approach :-(

.net TPL has a Complete() method http://msdn.microsoft.com/en-us/library/hh228601%28v=vs.110%29.aspx I can't find anything about re-opening.

Go allows channels to be closed https://gobyexample.com/closing-channels , does not flush items, and cannot reopen a channel https://groups.google.com/forum/#!topic/golang-nuts/e0jYSvJhPqA

This clojure library has produce-done https://github.com/martintrojer/pipejine . I couldn't find anything about re-opening, but I suspect a clojure library wouldn't go for that anyway.

Nobuyoshi Nakada wrote:

> (3) Shouldn't Queue#pop also raise an exception if the queue is empty and closed,
> instead of returning nil?

I'm not sure. nil allows for

```
while item = queue.deq
  ...
```

```
end
```

whereas StopIteration would work nicely with

```
loop do
  item = queue.deq
  ...
end
```

maybe both - queue.close(StopIteration). But that raises other questions - what to do when this happens:

```
queue.close
queue.close(RuntimeError.new 'queue is now closed')
queue.close(StopIteration)
```

But the parameter to queue.close would have to be stored anyway to know what to return from deq, so subsequent calls to close could check that the new parameter == the old parameter.

(4) What happens on another thread which is blocked at SizedQueue#push?

Thanks, I didn't think of that. I think the reason for Queue#push to raise an exception when the queue is closed is to signal that the programmer made an error. So following that logic, when the producer side calls queue.close and then continues to enq items, that's a programmer error.

Does that make sense? If so I'll update the patch to make SizedQueue#push behave like that.

#4 - 12/17/2014 05:44 PM - djellemah (John Anderson)

- File queue-close-2.diff added

I thought this specification would be more clear:

```
/*
 * Document-method: Queue#close
 * call-seq: close
 *
 * Closes the queue to producers. A closed queue cannot be re-opened.
 *
 * After the call to close completes, the following are true:
 *
 * - closed? will return true
 *
 * - calling enq/push/<< will raise an exception
 *
 * - calling deq/pop/shift will return an object from the queue as usual.
 *
 * - when empty? is true, deq(non_block=false) will not suspend and
 *   will return nil. deq(non_block=true) will raise an exception.
 *
 * And for SizedQueue, these will also be true:
 *
 * - each thread already suspended in enq at the time of the call
 *   to close will be allowed to push its object as usual.
 *
 * - empty? will be false when there are either objects in the queue, or
 *   producers which were suspended at the time of the call to close but whose
 *   objects are not yet in the queue. Therefore, it can be true (very
 *   briefly) that empty? == false && size == 0, since size returns the number
 *   of objects actually in the queue.
 */
```

An updated patch to implement that is attached. I've written some updated tests as well, but I've left those out of the patch for now.

I thought about (4) some more. What I've implemented in this patch was more difficult than throwing an exception as I suggested previously, but I think the semantics of this approach are somewhat less surprising.

#5 - 02/25/2015 07:56 PM - djellemah (John Anderson)

- File patch-25f99aef.diff added

Here is the full patch including tests and updated rdoc comments. diffed from current trunk 25f99aef.

#6 - 03/19/2015 11:56 AM - djellemah (John Anderson)

Another item for the survey - this is how Go channels implement close (and rendezvous)

https://docs.google.com/document/d/1yIAYmbvL3JxOKOjuCyon7JhW4cSv1wy5hC0ApeGMV9s/pub

#7 - 03/20/2015 06:15 PM - djellemah (John Anderson)

Closable queues in c++ in google-concurrency-library

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3533.html#closed_queues

http://code.google.com/p/google-concurrency-library/

#8 - 03/25/2015 07:31 PM - djellemah (John Anderson)

- File queue_benchmark.rb added

Some performance numbers, using the attached benchmark script:

```
$ ruby queue_benchmark.rb 100000
RUBY_DESCRIPTION: ruby 2.3.0dev (2015-03-25 trunk 50089) [x86_64-linux]
Queue#close: no
                                                 user     system      total        real
01 producer 01 consumer                      2.230000   0.110000   2.340000 (  2.219983)
01 producer 02 consumer                      2.360000   0.170000   2.530000 (  2.348708)
01 producer 99 consumer                      9.450000   5.290000  14.740000 ( 10.081818)
02 producer 01 consumer                      2.420000   0.080000   2.500000 (  2.348568)
99 producer 01 consumer                      6.850000   3.940000  10.790000 (  7.464203)


$ ruby queue_benchmark.rb 100000
RUBY_DESCRIPTION: ruby 2.3.0dev (2015-03-25 queue-close 50089) [x86_64-linux]
Queue#close: yes
                                                 user     system      total        real
01 producer 01 consumer                      2.380000   0.120000   2.500000 (  2.368862)
01 producer 02 consumer                      2.460000   0.170000   2.630000 (  2.460940)
01 producer 99 consumer                      9.420000   5.350000  14.770000 ( 10.075400)
02 producer 01 consumer                      2.970000   0.130000   3.100000 (  2.894214)
99 producer 01 consumer                      7.050000   4.100000  11.150000 (  7.676364)
```

#9 - 04/27/2015 07:28 PM - djellemah (John Anderson)

clojure's core.async has close! which implements the same semantics proposed by this issue.

https://clojure.github.io/core.async/#clojure.core.async/close%21

#10 - 08/22/2015 09:09 AM - ko1 (Koichi Sasada)

- Assignee set to ko1 (Koichi Sasada)

Thank you for your great survey. I want to introduce Queue#close in Ruby 2.3.

Just now I'm not sure it is okay to provide think Queue#close(token) API because there are no similar examples in Ruby.

The followings are summary of your survey.

language	API	deq from empty queue after close?
Java	N/A	
go	Close	return with indication https://golang.org/ref/spec#Close
C++	close()	return queue_op_status::closed (element is returned by reference)
closure	close!	return nil

Ruby's similar operation	What happen after read from empty stream?
File#read	return nil
File#read_nonblock()	raise EOFError
File#read_nonblock(exception: false)	return nil
File#gets	return nil

Options:

1. Queue#close(token)
2. Queue#close() and raise on deq from empty closed Queue
3. Queue#close() and return nil from empty closed Queue (raise by deq(nonblock=true))
4. Queue#close(exc) -> (2) if exc is not nil, (3) if exc is nil
5. Queue#close(exception: true/false) -> (2) if exception is true (specific exception, such as ClosedQueueError < StopIteration), (3) if exception is false
6. Queue#close() and provide Queue#deq(exception: false)

(3) is similar to IO's gets/read/...
(6) is similar to IO's read_nonblock.

I think (1) is over-spec. (4) should be nice than (1). But I like (5) because it is more simple.

#11 - 08/23/2015 07:19 AM - funny_falcon (Yura Sokolov)

You misread about Go channel:

> **Sending to or closing** a closed channel causes a **run-time panic**.
>
> (on empty channel) **receive** operations will return the **zero value** for the channel's type without blocking. The multi-valued receive operation returns a received value along with an **indication** of whether the channel is **closed**.

#12 - 08/26/2015 10:59 PM - ko1 (Koichi Sasada)

- Status changed from Open to Closed

Applied in changeset r51699.

- thread_tools.c: add Queue#close(exception=false) and SizedQueue#close(exception=false). [Feature #10600] Trying to deq from a closed empty queue return nil if exception parameter equals to false (default). If exception parameter is truthy, it raises ClosedQueueError (< StopIteration). ClosedQueueError inherits StopIteration so that you can write: loop{ e = q.deq; (using e) } Trying to close a closed queue raises ClosedQueueError. Blocking threads to wait deq for Queue and SizedQueue will be restarted immediately by returning nil (exception=false) or raising a ClosedQueueError (exception=true). Blocking threads to wait enq for SizedQueue will be restarted by raising a ClosedQueueError immediately. The above specification is not proposed specification, so that we need to continue discussion to conclude specification this method.
- test/thread/test_queue.rb: add tests originally written by John Anderson and modify detailed behavior.

#13 - 08/26/2015 11:19 PM - ko1 (Koichi Sasada)

- Status changed from Closed to Assigned

I committed r51699 to try Queue#close.

I changed proposed behavior:

- #close(token=nil) -> #close(exception=false) (variant of (5) in #10) because:
  - I feel strange that raising exception if token is Exception (I can't pass Exception objects with token)
  - Considering exception name is not valuable task. Only "ClosedQueueError" is enough. No need to worry about exception type.
- wake-up all blocking threads waiting enq for SizedQueue and raise ClosedQueueError because:
  - waiting threads can block eternally if no consumer threads deq a Queue.
  - It is simple rule to know: "nobody can not enq closed Queue". I think "waiting for enq" is BEFORE enq.

Could you try that?

Discussion:

- How about the above (committed) specification?
- ClosedQueueError inherits StopIteration, not ThreadError. Is it okay?
- "exception" optional parameter is reasonable or not? Should be "#close(exception: false)" or "#close!"?

BTW, I found that it is nice feature to synchronize starting multiple threads together.

```
synq = Queue.new
10.times{
  Thread.new{
    synq.pop #=> nil from closed Queue.
    # do something
  }
}
```

```
# do something initialization
synq.close
```

#14 - 08/27/2015 05:33 AM - nagachika (Tomoyuki Chikanaga)

Hello,
I'm interested in this topic.
I have some opinions about API design.

I'd like to specify the object to be returned by closed Queue#pop. Application could push nil to Queue as a significant value and.

And I think whether Queue#pop return nil(or something indicate the `EOQ') or raise exception should be determined by parameter of Queue#pop/deq.

If the behavior of Queue#pop is specified by Queue#close, you should know how the queue could be closed to write the code call Queue#pop, but it could be written by different programmers. And the worse the both could be happen.

How about adding keyword argument to Queue#pop,deq?

```
queue.pop(exception: false, eoq: nil) # raise ClosedQueue if exception is true, otherwise return eoq.
```

At last, if you can set counter to Queue#close *really close* the Queue, it easy to write multiple producer pattern. This is an advanced functionality and could be discussed on another ticket. How do you think?

ex)

```
def produce(q)
  while obj = get_something
    q.push(obj)
  end
  q.close
end
writers = 4
q = Queue.new(writers_count: writers)
writers.times { Thread.start { produce(q) } }
while obj = q.pop  # q.pop return nil after q.close was called 4 times
  # do something
end
```

#15 - 09/01/2015 09:35 AM - ko1 (Koichi Sasada)

> At last, if you can set counter to Queue#close really close the Queue, it easy to write multiple producer pattern. This is an advanced functionality and could be discussed on another ticket. How do you think?

I allow to close multiple times because IO#close also permits multiple close.

I can agree that close() should not have option and deq specify behavior.
Which is suitable default?

It is trivial concern, but keyword parameters for C methods are bit slow.
So that pop(keywords...) should be slower than without keywords. (but trivial)

#16 - 09/03/2015 01:08 PM - djellemah (John Anderson)

Sorry I didn't reply earlier, it's been a while since I checked this list.

I think ClosedQueueError < StopIteration makes sense. ThreadError (from other methods) is not related to ClosedQueueError, but I can't see if that is a problem.

I have some real-world code (because of db-connections, operations must be on separate threads. Might also be useful for Fibers?) which can now be simplified to something like this:

```
class NotificationActor
  def initialize
    @queue = SizedQueue.new 1
  end

  def stop
    @queue.close(exception: true)
  end
```

```
  def run
    consumer = Thread.new do
      begin
        loop do
          next_item = @queue.pop
          notify_listeners_of next_item
        end
      rescue
        # shut down as quickly as possible
        @queue.close(exception: true).clear
        raise
      end
    end

    loop do
      items_from_db {|item| @queue << item }
    end

  ensure
    # shut down as quickly as possible
    @queue.close(exception: true).clear
    # raise possible exceptions from consumer
    consumer.kill unless consumer.join(5)
  end
end
```

I'm not very happy with that design, but I think it is a reasonable real-world use of SizedQueue.

Re-doing that code, close(exception: true) caught me out twice. deq(non_block: true) normally catches me out too. Perhaps it's good that they both consistently catch me out ;-)

If it is necessary to support a non-nil close token, perhaps something like Queue.new(close_token: some_unique_object) would be better than close(some_unique_object).

When deq/pop takes parameters (like IO#read_nonblock) the code is clear, but for one instance of Queue those parameters will most likely be the same for every call.

Which makes me think maybe Queue.new(close_with_exception: true).

"'waiting for enq' is BEFORE enq" - yes, from the inside of the queue. From outside the queue they are part of the same operation. The idea behind close was to have a clean shut-down. If that still applies, maybe there needs to be another method for emergency shut-down. Or maybe queue.close.clear is sufficient for that?

#17 - 11/21/2015 12:29 AM - ko1 (Koichi Sasada)

I decide to reduce specification of Queue#close. For closed queues, deq returns nil.

No exception is raised for deq.
Other tokens are also not supported.

We can introduce them as new feature.

For Ruby 2.3 (or just now), Queue#close is only for shortcut of such common case.

```
consumer_threads = (1..3).map{
  Thread.new do
    while e = q.pop
      do_something e
    end
  end
}

q.push 1
q.push 2
3.times{
  q.push nil # terminater
}
```

We can write last 3 lines with:

`q.close`

I agree that it is reasonable to add options (raise exception, and so on) to Queue.new.
We can add this feature later.

#18 - 11/21/2015 12:32 AM - ko1 (Koichi Sasada)

- Status changed from Assigned to Closed

Applied in changeset r52691.

- thread_sync.c: reduce the specification of Queue#close.
  - Queue#close accepts no arguments.
  - deq'ing on closed queue returns nil, always. [Feature #10600]

- test/thread/test_queue.rb: catch up this fix.

Files

queue-close.diff	5.18 KB	12/15/2014	djellemah (John Anderson)
queue-close-2.diff	10.2 KB	12/17/2014	djellemah (John Anderson)
patch-25f99aef.diff	25.2 KB	02/25/2015	djellemah (John Anderson)
queue_benchmark.rb	2.95 KB	03/25/2015	djellemah (John Anderson)