Ruby - Feature #18965

Further Thread::Queue improvements

08/18/2022 02:46 PM - byroot (Jean Boussier)

Status:	Rejected
Priority:	Normal
Assignee:	
Target version:	

Description

Following the recent addition of a timeout parameter to Queue#pop, there are a handful of other improvements I'd like to make.

Batch insert

When using the queue for batch processing, it would be good to be able to push multiple elements at once:

Currently you have to call push repeatedly

```
items.each do |item|
  queue.push(item)
end
```

That's wasteful because on each call we check wether the queue is closed, try to wakeup blocked threads, etc.

It would be much better if you could do:

```
queue.concat(items)
```

With of course both nonblock and timeout support.

Then there's the question of how SizedQueue would behave if it's not full, but still doesn't have space for all the elements. e.g.

```
queue = SizedQueue.new(10)
queue.concat(6.times.to_a)
queue.concat(6.times.to_a) # Block until there is 6 free slots?
```

I think the simplest would be to wait for enough space to append the entire set, because combined with a timeout, it would be awkward if only part of the array was concatenated.

Batch pop

Similarly, sometimes the consumer of a queue is capable of batching, and right now it's not efficient:

```
loop do
  items = [queue.pop]
  begin
    99.times do
      items << queue.pop(true) # true is for nonblock
    end
  rescue ThreadError # empty queue
  end
  process_items(items)
end
```

It would be much more efficient if pop accepted a count parameter:

```
loop do
  items = queue.pop(count: 100)
  process_items(items)
end
```

The behavior would be:

- Block if the queue is empty
- If it's not empty, return **up to** count items (Just like Array#pop)

Non blocking mode, without exception

As shown above, the current nonblock parameter is a bit awkward, because:

- It raises an exception, which is very expensive for a construct often used in "low level" code.
- The exception is ThreadError, so you may have to match the error message for "queue empty", to make sure it doesn't come from a Mutex issue or something like that.

I believe that we could introduce a keyword argument:

```
Queue.new.pop(nonblock: true) # => nil
```

Related issues:

Related to Ruby - Feature #18982: Add an `exception: false` argument for Queu...	**Closed**

History

#1 - 08/18/2022 05:06 PM - p8 (Petrik de Heus)

Hi Jean,
Should that second code example use items instead of item?

```
queue.concat(items)
```

#2 - 08/18/2022 05:56 PM - byroot (Jean Boussier)

- Description updated

@p8, indeed. Thanks for the heads up.

#3 - 08/18/2022 06:30 PM - chrisseaton (Chris Seaton)

I was going to comment that adding or removing multiple items from a queue is likely not great for implementation, as we'd need a lock to make that atomic (or some good ideas of how to do it otherwise.) But then I looked at TruffleRuby and even we're using a lock anyway, so nothing here is a problem.

We should carefully specify how atomic this operation is supposed to be though!

#4 - 08/18/2022 06:41 PM - Eregon (Benoit Daloze)

> Then there's the question of how SizedQueue would behave if it's not full, but still doesn't have space for all the elements. e.g.
> I think the simplest would be to wait for enough space to append the entire set, because combined with a timeout, it would be awkward if only part of the array was concatenated.

That would mean basically ignoring the timeout, not great.
And other options like returning the number of elements pushed or mutating the array don't seem too nice either.

I think for these batch push/pop, they should be motivated with benchmarks (ideally based on some real use-case) and a proof-of-concept PR.
IMHO it makes sense to add them only if there is a significant performance gain.

Also this can cause contention and e.g. starve the opposite operation, e.g. a big batch push of 1000 elements and other threads trying to pop are all blocked while pushing these elements, not good for latency/contention.

> Non blocking mode, without exception

Agreed we should add that.

The kwarg seems a bit confusing given the existing positional arg (Queue#pop(non_block=false)).
OTOH nonblock is kind of the established term for this (e.g., read_nonblock).
Is there any core method currently with a nonblock keyword argument?

Why not exception: false which seems more standard and established, i.e., queue.pop(true, exception: false)? Too verbose/inconvenient?
Some related discussion in https://bugs.ruby-lang.org/issues/18774#note-11

Could also be a new method like pop_nonblock (like read_nonblock), not sure if a good idea but putting it out there for thoughts.

#5 - 08/18/2022 06:59 PM - Eregon (Benoit Daloze)

chrisseaton (Chris Seaton) wrote in #note-3:

> I was going to comment that adding or removing multiple items from a queue is likely not great for implementation, as we'd need a lock to make that atomic (or some good ideas of how to do it otherwise.) But then I looked at TruffleRuby and even we're using a lock anyway, so nothing here is a problem.

It could be though.

At some point TruffleRuby used Java's LinkedBlockingQueue (removed in this PR because it's hard to implement Queue#num_waiting without poking in JVM internals) and JRuby currently uses a variant of it, which are non-blocking queue implementations, so they (typically) cannot do multiple operations (push, pop, ...) atomically.
(there are also trade-offs with a non-blocking queue as it might be slower in single-threaded/low contention cases, e.g. https://github.com/oracle/truffleruby/issues/595)

Java's LinkedBlockingQueue also doesn't seem to implement addAll() so it's just the default implementation adding one by one, hence addAll() is not atomic and the added elements might be interleaved with another thread's pushed elements.
That might be an indication a batch push/pop is not that big a gain or too problematic for contention.

#6 - 08/18/2022 07:59 PM - byroot (Jean Boussier)

> That would mean basically ignoring the timeout, not great.

Maybe my writing wasn't clear. I don't suggest to ignore the timeout, but to either append all items or none. If there isn't enough space available, wait for as long as the timeout allows you to.

> I think for these batch push/pop, they should be motivated with benchmarks (ideally based on some real use-case) and a proof-of-concept PR.

Like the previous changes I requested on queue, this is motivated by a real world use case, basically I'd like to use a SizedQueue in this code: https://github.com/Shopify/statsd-instrument/blob/da8a74c7fbe8fdd421aa1df7eda2a996bc7c3f11/lib/statsd/instrument/batched_udp_sink.rb#L93-L123

This library has a background thread that monitors a queue, everytime the queue has X items or that Y seconds have passed, it flushes the queue and emit statsd events.

I can try implementing the suggested feature and then benchmark that code using it.

> The kwarg seems a bit confusing given the existing positional arg (Queue#pop(non_block=false)).

Indeed, in my mind the positional argument would be somewhat deprecated.

> Is there any core method currently with a nonblock keyword argument?

I don't think so.

> Why not exception: false which seems more standard and established, i.e., queue.pop(true, exception: false)? Too verbose/inconvenient?

I think the main reason is that I don't like that positional argument. But yes exception: false would be consistent with IO#read_nonblock, except that if passed alone it does nothing, which is awkward.

> so it's just the default implementation adding one by one, hence addAll() is not atomic

I don't think that's really a problem, at least not for the use cases I envision, of a background thread collecting work and processing it in batches. Worst case if it's not atomic, the background thread will get the first element, and then get the rest on its next pop.

#7 - 08/19/2022 11:33 AM - byroot (Jean Boussier)

Also note that I did a single ticket for 3 somewhat related features, but if concat (aka batch push) is deemed undesirable, is doesn't prevent from going ahead with the other two suggested features.

#8 - 08/19/2022 03:43 PM - byroot (Jean Boussier)

So I implemented pop(:count) https://github.com/ruby/ruby/pull/6257

And benchmarked it against my somewhat realistic benchmark: https://github.com/Shopify/statsd-instrument/pull/315

The TL;DR; is that this benchmark is mostly bound on UDPSocket#send and some string concatenation, Queue#pop(true) only account for 0.3% of runtime, so pop(count:) bringing that share down to 0.08% doesn't make any visible difference on the overall benchmark.

Whether or not that improvement would show on different (non-micro) benchmark is unclear.

#9 - 08/20/2022 11:13 AM - Eregon (Benoit Daloze)

Thank you for the benchmark.
Given the results, I think it's currently not worth it to add batch push/pop, because it does not seem to improve performance much over a loop of single push/pop in realistic workloads and the semantics are quite a bit more complicated.
Also depending on the Queue/SizedQueue implementation (typical for non-blocking queues) there might simply not be a way to do batch push/pop, making the performance benefit actually zero for those implementations (well, slightly less Ruby calls but JITs are good at removing the cost of that).

I'd suggest to focus on Queue.new.pop(nonblock: true) # => nil.
Using the keyword as a way to deprecate the positional argument makes sense to me, so I agree with pop(nonblock: true/false).

#10 - 08/20/2022 12:48 PM - byroot (Jean Boussier)

pop(nonblock: true) that wouldn't raise definitely make sense to me, so I'll for sure propose it at the next meeting.

> Also depending on the Queue/SizedQueue implementation (typical for non-blocking queues) there might simply not be a way to do batch push/pop,

Is it true for both push and pop? I can see the issue for push, but no so much for pop.

I no longer think a batch push is necessary, so I'll remove that part of the proposal, but I still think pop(count:) makes the calling code much much simpler, so I'd still like to have it even if it doesn't improve performance significantly.

#11 - 08/21/2022 12:13 PM - Eregon (Benoit Daloze)

byroot (Jean Boussier) wrote in #note-10:

> Is it true for both push and pop? I can see the issue for push, but no so much for pop.

Interesting, for LinkedBlockingQueue there is java.util.concurrent.BlockingQueue#drainTo(java.util.Collection<? super E> output, int maxElements) (semantics are remove up to maxElements, don't wait for more elements).
But LinkedBlockingQueue is actually not a non-blocking queue (it uses 2 locks).

ConcurrentLinkedQueue is a non-blocking queue and more what I was thinking about for non-blocking queue implementations.
That doesn't have a batch pop, and is literally just CAS operations for both push and pop.
But that's probably not directly usable to implement Ruby's Queue since it doesn't have a blocking pop which Queue#pop needs.

I think there can be queue implementations in between, which use CAS(node) for push/pop, but if it fails then goes to a lock or similar to wait.
In such a case, it's not possible to implement batch push/pop atomically, or it would force to give up on the CAS fast path and take a lock always (a lock can use CAS internally but basically it's extra work, indirections and most importantly contention).
It's still possible to implement batch pop as N.times { queue.pop } of course (with no performance benefit).

I think the problem is mostly semantics, if there is a batch pop, it might lead to people to think it removes N elements atomically or any better than N.times { queue.pop }, but in fact it might not (atomic or better perf).
And if a single Ruby implementation actually has atomic semantics there, people might rely on it, just like they rely on Set being ordered (even though it was never documented).
And therefore that would basically prevent some queue implementations to be used for compatibility (and e.g. force all Ruby Queue impls to be single-lock or maybe dual-lock depending on the atomicity guarantees).
Things like linearizability just don't work for batch push/pop.
So I think it's better to keep single push/pop, because those semantics are well-defined and honored by all queue implementations.
If the user does N.times { queue.pop }, then the semantics are very clear.

#12 - 08/26/2022 03:14 PM - byroot (Jean Boussier)

> Why not exception: false which seems more standard and established

So after working more on using a SizedQueue in statsd-instrument, I now think exception: false might actually be better.

The reason is that SizedQueue#push(item, true) can raise in two cases:

- If the SizedQueue is full.
- If the SizedQueue is closed.

So exception: false would be "two birds one stone", and would be consistent with the various IO methods.

#13 - 08/29/2022 10:21 AM - byroot (Jean Boussier)

- Status changed from Open to Rejected

Closing in favor of [Feature #18982], which propose exception: false.

#14 - 08/29/2022 10:21 AM - byroot (Jean Boussier)

- Related to Feature #18982: Add an `exception: false` argument for Queue#push, Queue#pop, SizedQueue#push and SizedQueue#pop added