Ruby - Bug #21210

IO::Buffer gets invalidated on GC compaction

04/01/2025 04:39 PM - hanazuki (Kasumi Hanazuki)

Status:	Closed		
Priority:	Normal		
Assignee:	ioquatix (Samuel Williams)		
Target version:			
ruby -v:	ruby 3.5.0dev (2025-04-01T16:11:01Z master 30e5e7c005) +PRISM [x86_64-linux]	**Backport:**	3.1: DONTNEED, 3.2: DONTNEED, 3.3: DONTNEED, 3.4: DONTNEED

Description

6012145299cfa4ab561360c78710c7f2941a7e9d implemented compaction for IO::Buffer.
It looks like this doesn't work well with an IO::Buffer that shares memory region with a String object.
I think the problem is that an IO::Buffer holds the raw pointer to the String content, and now the content can be moved by GC when the String is embedded.

```
str = +"hello"
buf = IO::Buffer.for(str)
p buf.valid?

GC.verify_compaction_references(expand_heap: true, toward: :empty)

p buf.valid?  #=> should be true
```

This example should print two trues. Actually:

```
% ./ruby -v --disable-gems test.rb
ruby 3.5.0dev (2025-04-01T16:11:01Z master 30e5e7c005) +PRISM [x86_64-linux]
true
false
```

Associated revisions

Revision 8aac19d5 - 06/17/2025 05:57 AM - hanazuki (Kasumi Hanazuki)

io_buffer: Reimplement dcompact for IO::Buffer

The source field in IO::Buffer can have a String or an IO::Buffer
object, if not nil.

- When the source is a String object. The base field points to the
  memory location of the String content, which can be embedded in
  RSTRING, and in that case, GC compaction can move the memory region
  along with the String object.

  Thus, IO::Buffer needs to pin the source object to prevent base
  pointer from becoming invalid.

- When the source is an IO::Buffer, then base is a pointer to a
  malloced or mmapped memory region, managed by the source IO::Buffer.
  In this case, we don't need to pin the source IO::Buffer object,
  since the referred memory region won't get moved by GC.

Closes: [Bug #21210]

History

#1 - 04/01/2025 05:17 PM - byroot (Jean Boussier)

- Backport changed from 3.1: UNKNOWN, 3.2: UNKNOWN, 3.3: UNKNOWN, 3.4: UNKNOWN to 3.1: DONTNEED, 3.2: DONTNEED, 3.3: DONTNEED, 3.4: DONTNEED

#2 - 04/01/2025 05:28 PM - hanazuki (Kasumi Hanazuki)

The source field in a struct rb_io_buffer can have a String or an IO::Buffer if not nil.

When source is a String, we need to pin the object (rb_str_locktmp does not keep the embedded String content from being moved by GC).
When source is an IO::Buffer, I believe it can be moved, because the base pointer refers to a malloced or mmapped memory region, which does not get compacted by GC.

#3 - 04/01/2025 08:10 PM - hanazuki (Kasumi Hanazuki)

Patch: https://github.com/ruby/ruby/pull/13033

#4 - 04/01/2025 09:56 PM - eightbitraptor (Matt V-H)

I checked with the following test.rb that this patch does fixes the Buffer validity after compaction.

```
str = +"hello"

str_buf = IO::Buffer.for(str)

buf = IO::Buffer.new(128)
slice = buf.slice(8, 32)

GC.verify_compaction_references(expand_heap: true, toward: :empty)

p str_buf.valid?
p buf.valid?
p slice.valid?
```

Instead of pinning the source string, did you consider allowing the string to move by calculating the base offset, then moving the source string and updating the base pointer using rb_gc_location(buffer->source) and the calculated offset? Would this work?

#5 - 04/01/2025 11:41 PM - alanwu (Alan Wu)

Another option that maintains validity across movement (untested):

```
diff --git a/io_buffer.c b/io_buffer.c
index 0534999319..13102b561d 100644
--- a/io_buffer.c
+++ b/io_buffer.c
@@ -570,7 +570,7 @@ rb_io_buffer_type_for(VALUE klass, VALUE string)
     }
     else {
         // This internally returns the source string if it's already frozen.
-        string = rb_str_tmp_frozen_acquire(string);
+        string = rb_str_tmp_frozen_no_embed_acquire(string);
         return io_buffer_for_make_instance(klass, string, RB_IO_BUFFER_READONLY);
     }
 }
```

Looks like the mutable string buffer code paths pin using rb_str_locktmp().

#6 - 04/02/2025 01:39 AM - hanazuki (Kasumi Hanazuki)

eightbitraptor (Matt V-H) wrote in #note-4:

> Instead of pinning the source string, did you consider allowing the string to move by calculating the base offset, then moving the source string and updating the base pointer using rb_gc_location(buffer->source) and the calculated offset?

I think that approach can be possible if we check that the buffer is not locked before allowing the String to move. The base pointer may have been passed to a syscall or C library call, in which case we cannot move it. Extensions using the raw pointer guard the section with rb_io_buffer_lock and rb_io_buffer_unlock. So we can check RB_IO_BUFFER_LOCKED flag to see if such a syscall is running.

Besides, there is another (unreported?) problem to be fixed. In the current implementation, the RB_IO_BUFFER_LOCKED flag of an IO::Buffer is not correctly synced among its slices, which share the same memory region.

```
str = +"hello"
buf = IO::Buffer.for(str)
slice = buf.slice

slice.locked do
  p buf.locked?  #=> false
  # Moving or freeing `buf` here may cause something bad, as we expect the base pointer of `slice` is pinned.
end
```

I will file this as another ticket. #=> #21212

#7 - 04/02/2025 02:40 AM - hanazuki (Kasumi Hanazuki)

alanwu (Alan Wu) wrote in #note-5:

> Another option that maintains validity across movement (untested):
>
> ```
> diff --git a/io_buffer.c b/io_buffer.c
> index 0534999319..13102b561d 100644
> --- a/io_buffer.c
> +++ b/io_buffer.c
> @@ -570,7 +570,7 @@ rb_io_buffer_type_for(VALUE klass, VALUE string)
>      }
>      else {
>          // This internally returns the source string if it's already frozen.
> -        string = rb_str_tmp_frozen_acquire(string);
> +        string = rb_str_tmp_frozen_no_embed_acquire(string);
>          return io_buffer_for_make_instance(klass, string, RB_IO_BUFFER_READONLY);
>      }
>  }
> ```

So this will copy the embedded String content to a malloc'ed memory, right?

The default GC embeds up to 640 bytes minus RSTRING header (IIUC). I think we need to evaluate the performance impact of adding such size of copies.

```
packet = "x" * 600
buf = IO::Buffer.for(packet)
buf.get_value(:U8, 32)
```

> Looks like the mutable string buffer code paths pin using rb_str_locktmp().

Ah, yes. rb_str_locktmp was not relevant to IO::Buffer.for without a block.

In case of IO::Buffer.for with a block, the source String is referred to by a C stack variable, and so the object will not be moved.

#8 - 05/28/2025 06:16 PM - ioquatix (Samuel Williams)

- Assignee set to ioquatix (Samuel Williams)

#9 - 06/17/2025 05:57 AM - hanazuki (Kasumi Hanazuki)

- Status changed from Open to Closed

Applied in changeset git|8aac19d5987150cf5c45fee73c7a949ca472f488.

---

io_buffer: Reimplement dcompact for IO::Buffer

The source field in IO::Buffer can have a String or an IO::Buffer
object, if not nil.

- When the source is a String object. The base field points to the
  memory location of the String content, which can be embedded in
  RSTRING, and in that case, GC compaction can move the memory region
  along with the String object.

  Thus, IO::Buffer needs to pin the source object to prevent base
  pointer from becoming invalid.

- When the source is an IO::Buffer, then base is a pointer to a
  malloced or mmapped memory region, managed by the source IO::Buffer.
  In this case, we don't need to pin the source IO::Buffer object,
  since the referred memory region won't get moved by GC.

Closes: [Bug #21210]

#10 - 06/17/2025 07:10 AM - Eregon (Benoit Daloze)

alanwu (Alan Wu) wrote in #note-5:

> Looks like the mutable string buffer code paths pin using rb_str_locktmp().

Does rb_str_locktmp() pin? I couldn't see anything like that from the usage of the flag it sets.

#11 - 06/18/2025 06:39 AM - hanazuki (Kasumi Hanazuki)

I think @alanwu (Alan Wu) means the mutable buffer path (IO::Buffer.for w/ block) uses rb_str_locktmp to pin the String's malloc'ed content memory (not RString), while the immutable buffer path (IO::Buffer.for w/o block) utilizes CoW to obtain a frozen copy of the given String, pointing out that my initial explanation mentioning rb_str_locktmp was wrong.