Lockless Ring Buffer Design

   Author: Steven Rostedt <srostedt@redhat.com>

Reviewers: Mathieu Desnoyers, Huang Ying, Hidetoshi Seto,
             and Frederic Weisbecker.

Written for: 2.6.31

Terminology used in this Document

tail - where new writes happen in the ring buffer.

head - where new reads happen in the ring buffer.

producer - the task that writes into the ring buffer (same as writer)

writer - same as producer

consumer - the task that reads from the buffer (same as reader)

reader - same as consumer.

reader_page - A page outside the ring buffer used solely (for the most part) by the reader.

head_page - a pointer to the page that the reader will use next

tail_page - a pointer to the page that will be written to next

commit_page - a pointer to the page with the last finished non-nested write.

cmpxchg - hardware-assisted atomic transaction that performs the following:

```
   A = B iff previous A == C

   R = cmpxchg(A, C, B) is saying that we replace A with B if and only if
      current A is equal to C, and we put the old (current) A into R

   R gets the previous A regardless if A is updated with B or not.

   To see if the update was successful a compare of R == C may be used.
```

The Generic Ring Buffer

The ring buffer can be used in either an overwrite mode or in producer/consumer mode.

Producer/consumer mode is where if the producer were to fill up the buffer before the consumer could free up anything, the producer will stop writing to the buffer. This will lose most recent events.

Overwrite mode is where if the producer were to fill up the buffer
before the consumer could free up anything, the producer will
overwrite the older data. This will lose the oldest events.

No two writers can write at the same time (on the same per-cpu buffer),
but a writer may interrupt another writer, but it must finish writing
before the previous writer may continue. This is very important to the
algorithm. The writers act like a "stack". The way interrupts works
enforces this behavior.

```
  writer1 start
     <preempted> writer2 start
         <preempted> writer3 start
                     writer3 finishes
                 writer2 finishes
  writer1 finishes
```

This is very much like a writer being preempted by an interrupt and
the interrupt doing a write as well.

Readers can happen at any time. But no two readers may run at the
same time, nor can a reader preempt/interrupt another reader. A reader
cannot preempt/interrupt a writer, but it may read/consume from the
buffer at the same time as a writer is writing, but the reader must be
on another processor to do so. A reader may read on its own processor
and can be preempted by a writer.

A writer can preempt a reader, but a reader cannot preempt a writer.
But a reader can read the buffer at the same time (on another processor)
as a writer.

The ring buffer is made up of a list of pages held together by a linked list.

At initialization a reader page is allocated for the reader that is not
part of the ring buffer.

The head_page, tail_page and commit_page are all initialized to point
to the same page.

The reader page is initialized to have its next pointer pointing to
the head page, and its previous pointer pointing to a page before
the head page.

The reader has its own page to use. At start up time, this page is
allocated but is not attached to the list. When the reader wants
to read from the buffer, if its page is empty (like it is on start-up),
it will swap its page with the head_page. The old reader page will
become part of the ring buffer and the head_page will be removed.
The page after the inserted page (old reader_page) will become the
new head page.

Once the new page is given to the reader, the reader could do what
it wants with it, as long as a writer has left that page.

A sample of how the reader page is swapped: Note this does not
show the head page in the buffer, it is for demonstrating a swap
only.

```
  +------+
  |reader|          RING BUFFER
  |page  |
  +------+
                  +---+   +---+   +---+
                  |   |-->|   |-->|   |
                  |   |<--|   |<--|   |
                  +---+   +---+   +---+
                   ^ |             ^ |
                   | +-------------+ |
                   +-----------------+


  +------+
  |reader|          RING BUFFER
  |page  |-------------------+
  +------+                   v
    |             +---+   +---+   +---+
    |             |   |-->|   |-->|   |
    |             |   |<--|   |<--|   |<-+
    |             +---+   +---+   +---+  |
    |              ^ |             ^ |   |
    |              | +-------------+ |   |
    |              +-----------------+   |
    +------------------------------------+

  +------+
  |reader|          RING BUFFER
  |page  |-------------------+
  +------+ <---------------+ v
    |  ^          +---+   +---+   +---+
    |  |          |   |-->|   |-->|   |
    |  |          |   |   |   |<--|   |<-+
    |  |          +---+   +---+   +---+  |
    |  |             |             ^ |   |
    |  |             +-------------+ |   |
    |  +-----------------------------+   |
    +------------------------------------+

  +------+
  |buffer|          RING BUFFER
  |page  |-------------------+
  +------+ <---------------+ v
    |  ^          +---+   +---+   +---+
    |  |          |   |   |   |-->|   |
    |  |  New     |   |   |   |<--|   |<-+
    |  | Reader   +---+   +---+   +---+  |
    |  |  page ----^                 |   |
    |  |                             |   |
    |  +-----------------------------+   |
    +------------------------------------+
```

It is possible that the page swapped is the commit page and the tail page,
if what is in the ring buffer is less than what is held in a buffer page.

```
          reader page    commit page   tail page
              |              |             |
              v              |             |
             +---+           |             |
             |   |<----------+             |
             |   |<------------------------+
             |   |------+
             +---+      |
                        |
                        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |--->|   |--->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

This case is still valid for this algorithm.
When the writer leaves the page, it simply goes into the ring buffer
since the reader page still points to the next location in the ring
buffer.

The main pointers:

```
  reader page - The page used solely by the reader and is not part
                of the ring buffer (may be swapped in)

  head page - the next page in the ring buffer that will be swapped
              with the reader page.

  tail page - the page where the next write will take place.

  commit page - the page that last finished a write.
```

The commit page only is updated by the outermost writer in the
writer stack. A writer that preempts another writer will not move the
commit page.

When data is written into the ring buffer, a position is reserved
in the ring buffer and passed back to the writer. When the writer
is finished writing data into that position, it commits the write.

Another write (or a read) may take place at anytime during this
transaction. If another write happens it must finish before continuing
with the previous write.

```
   Write reserve:

       Buffer page
```

```
     +---------+
     |written  |
     +---------+  <--- given back to writer (current commit)
     |reserved |
     +---------+ <--- tail pointer
     | empty   |
     +---------+

  Write commit:

      Buffer page
     +---------+
     |written  |
     +---------+
     |written  |
     +---------+  <--- next positon for write (current commit)
     | empty   |
     +---------+


If a write happens after the first reserve:

      Buffer page
     +---------+
     |written  |
     +---------+  <-- current commit
     |reserved |
     +---------+  <--- given back to second writer
     |reserved |
     +---------+ <--- tail pointer

 After second writer commits:


      Buffer page
     +---------+
     |written  |
     +---------+  <--(last full commit)
     |reserved |
     +---------+
     |pending  |
     |commit   |
     +---------+ <--- tail pointer

 When the first writer commits:

      Buffer page
     +---------+
     |written  |
     +---------+
     |written  |
     +---------+
     |written  |
     +---------+  <--(last full commit and tail pointer)
```

The commit pointer points to the last write location that was
committed without preempting another write. When a write that
preempted another write is committed, it only becomes a pending commit
and will not be a full commit until all writes have been committed.

The commit page points to the page that has the last full commit.
The tail page points to the page with the last write (before
committing).

The tail page is always equal to or after the commit page. It may
be several pages ahead. If the tail page catches up to the commit
page then no more writes may take place (regardless of the mode
of the ring buffer: overwrite and produce/consumer).

The order of pages is:

```
 head page
 commit page
 tail page
```

Possible scenario:

```
                             tail page
  head page         commit page  |
      |                 |        |
      v                 v        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |--->|   |--->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

There is a special case that the head page is after either the commit page
and possibly the tail page. That is when the commit (and tail) page has been
swapped with the reader page. This is because the head page is always
part of the ring buffer, but the reader page is not. Whenever there
has been less than a full page that has been committed inside the ring buffer,
and a reader swaps out a page, it will be swapping out the commit page.

```
          reader page    commit page   tail page
              |              |             |
              v              |             |
             +---+           |             |
             |   |<----------+             |
             |   |<------------------------+
             |   |------+
             +---+      |
                        |
                        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |--->|   |--->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
                        ^
                        |
                    head page
```

In this case, the head page will not move when the tail and commit
move back into the ring buffer.

The reader cannot swap a page into the ring buffer if the commit page
is still on that page. If the read meets the last commit (real commit
not pending or reserved), then there is nothing more to read.
The buffer is considered empty until another full commit finishes.

When the tail meets the head page, if the buffer is in overwrite mode,
the head page will be pushed ahead one. If the buffer is in producer/consumer
mode, the write will fail.

Overwrite mode:

```
            tail page
               |
               v
    +---+    +---+    +---+    +---+
<---|   |--->|   |--->|   |--->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
                        ^
                        |
                    head page


            tail page
               |
               v
    +---+    +---+    +---+    +---+
<---|   |--->|   |--->|   |--->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
                                 ^
                                 |
                             head page


                    tail page
                        |
                        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |--->|   |--->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
                                 ^
                                 |
                             head page
```

Note, the reader page will still point to the previous head page.
But when a swap takes place, it will use the most recent head page.

Making the Ring Buffer Lockless:

```
The main idea behind the lockless algorithm is to combine the moving
of the head_page pointer with the swapping of pages with the reader.
State flags are placed inside the pointer to the page. To do this,
each page must be aligned in memory by 4 bytes. This will allow the 2
least significant bits of the address to be used as flags, since
they will always be zero for the address. To get the address,
simply mask out the flags.

  MASK = ~3

  address & MASK

Two flags will be kept by these two bits:

   HEADER - the page being pointed to is a head page

   UPDATE - the page being pointed to is being updated by a writer
          and was or is about to be a head page.


          reader page
              |
              v
             +---+
             |   |------+
             +---+      |
                        |
                        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-H->|   |--->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+


The above pointer "-H->" would have the HEADER flag set. That is
the next page is the next page to be swapped out by the reader.
This pointer means the next page is the head page.

When the tail page meets the head pointer, it will use cmpxchg to
change the pointer to the UPDATE state:


            tail page
               |
               v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-H->|   |--->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+

            tail page
               |
               v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-U->|   |--->|   |--->
```

```
--->|    |<---|    |<---|    |<---|    |<---
    +----+    +----+    +----+    +----+
```

"-U->" represents a pointer in the UPDATE state.

Any access to the reader will need to take some sort of lock to serialize
the readers. But the writers will never take a lock to write to the
ring buffer. This means we only need to worry about a single reader,
and writes only preempt in "stack" formation.

When the reader tries to swap the page with the ring buffer, it
will also use cmpxchg. If the flag bit in the pointer to the
head page does not have the HEADER flag set, the compare will fail
and the reader will need to look for the new head page and try again.
Note, the flags UPDATE and HEADER are never set at the same time.

The reader swaps the reader page as follows:

```
  +-------+
  |reader|          RING BUFFER
  |page   |
  +-------+

                  +----+    +----+    +----+
                  |    |--->|    |--->|    |
                  |    |<---|    |<---|    |
                  +----+    +----+    +----+
                   ^ |                ^ |
                   | +----------------+ |
                   +-----H--------------+
```

The reader sets the reader page next pointer as HEADER to the page after
the head page.

```
  +-------+
  |reader|          RING BUFFER
  |page   |-------H-----------+
  +-------+                   v
    |             +----+    +----+    +----+
    |             |    |--->|    |--->|    |
    |             |    |<---|    |<---|    |<-+
    |             +----+    +----+    +----+   |
    |              ^ |                ^ |      |
    |              | +----------------+ |      |
    |              +-----H--------------+      |
    +-----------------------------------------+
```

It does a cmpxchg with the pointer to the previous head page to make it
point to the reader page. Note that the new pointer does not have the HEADER
flag set. This action atomically moves the head page forward.

```
  +-------+
  |reader|          RING BUFFER
  |page   |-------H-----------+
  +-------+                   v
    |  ^          +----+   +----+   +----+
    |  |          |   |-->|   |-->|   |
    |  |          |   |<--|   |<--|   |<-+
    |  |          +---+   +---+   +---+  |
    |  |             |             ^ |   |
    |  |             +-------------+ |   |
    |  +-----------------------------+   |
    +------------------------------------+
```

After the new head page is set, the previous pointer of the head page is
updated to the reader page.

```
  +------+
  |reader|          RING BUFFER
  |page  |-------H-----------+
  +------+ <---------------+ v
    |  ^          +---+   +---+   +---+
    |  |          |   |-->|   |-->|   |
    |  |          |   |   |   |<--|   |<-+
    |  |          +---+   +---+   +---+  |
    |  |             |             ^ |   |
    |  |             +-------------+ |   |
    |  +-----------------------------+   |
    +------------------------------------+

  +------+
  |buffer|          RING BUFFER
  |page  |-------H-----------+  <--- New head page
  +------+ <---------------+ v
    |  ^          +---+   +---+   +---+
    |  |          |   |   |   |-->|   |
    |  |  New     |   |   |   |<--|   |<-+
    |  | Reader   +---+   +---+   +---+  |
    |  |  page ----^                 |   |
    |  |                             |   |
    |  +-----------------------------+   |
    +------------------------------------+
```

Another important point: The page that the reader page points back to
by its previous pointer (the one that now points to the new head page)
never points back to the reader page. That is because the reader page is
not part of the ring buffer. Traversing the ring buffer via the next pointers
will always stay in the ring buffer. Traversing the ring buffer via the
prev pointers may not.

Note, the way to determine a reader page is simply by examining the previous
pointer of the page. If the next pointer of the previous page does not
point back to the original page, then the original page is a reader page:

```
             +---------+
             | reader  | next   +-----+
             |  page   |------->|     |<====== (buffer page)
             +---------+        +-----+
                 |                |  ^
                 |                v  | next
            prev |              +-----+
                 +------------->|    |
                                +----+
```

The way the head page moves forward:

When the tail page meets the head page and the buffer is in overwrite mode
and more writes take place, the head page must be moved forward before the
writer may move the tail page. The way this is done is that the writer
performs a cmpxchg to convert the pointer to the head page from the HEADER
flag to have the UPDATE flag set. Once this is done, the reader will
not be able to swap the head page from the buffer, nor will it be able to
move the head page, until the writer is finished with the move.

This eliminates any races that the reader can have on the writer. The reader
must spin, and this is why the reader cannot preempt the writer.

```
            tail page
               |
               v
    +----+    +----+    +----+    +----+
<---|    |--->|    |-H->|    |--->|    |--->
--->|    |<---|    |<---|    |<---|    |<---
    +----+    +----+    +----+    +----+

            tail page
               |
               v
    +----+    +----+    +----+    +----+
<---|    |--->|    |-U->|    |--->|    |--->
--->|    |<---|    |<---|    |<---|    |<---
    +----+    +----+    +----+    +----+
```

The following page will be made into the new head page.

```
           tail page
               |
               v
    +----+    +----+    +----+    +----+
<---|    |--->|    |-U->|    |-H->|    |--->
--->|    |<---|    |<---|    |<---|    |<---
    +----+    +----+    +----+    +----+
```

After the new head page has been set, we can set the old head page
pointer back to NORMAL.

```
           tail page
               |
               v
    +----+    +----+    +----+    +----+
<---|    |--->|    |--->|    |-H->|    |--->
--->|    |<---|    |<---|    |<---|    |<---
    +----+    +----+    +----+    +----+
```

After the head page has been moved, the tail page may now move forward.

```
                    tail page
```

```
                        |
                        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |--->|   |-H->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

The above are the trivial updates. Now for the more complex scenarios.

As stated before, if enough writes preempt the first write, the
tail page may make it all the way around the buffer and meet the commit
page. At this time, we must start dropping writes (usually with some kind
of warning to the user). But what happens if the commit was still on the
reader page? The commit page is not part of the ring buffer. The tail page
must account for this.

```
          reader page    commit page
              |              |
              v              |
             +---+           |
             |   |<----------+
             |   |
             |   |------+
             +---+      |
                        |
                        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-H->|   |--->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
               ^
               |
           tail page
```

If the tail page were to simply push the head page forward, the commit when
leaving the reader page would not be pointing to the correct page.

The solution to this is to test if the commit page is on the reader page
before pushing the head page. If it is, then it can be assumed that the
tail page wrapped the buffer, and we must drop new writes.

This is not a race condition, because the commit page can only be moved
by the outermost writer (the writer that was preempted).
This means that the commit will not move while a writer is moving the
tail page. The reader cannot swap the reader page if it is also being
used as the commit page. The reader can simply check that the commit
is off the reader page. Once the commit page leaves the reader page
it will never go back on it unless a reader does another swap with the
buffer page that is also the commit page.

Nested writes

In the pushing forward of the tail page we must first push forward
the head page if the head page is the next page. If the head page
is not the next page, the tail page is simply updated with a cmpxchg.

Only writers move the tail page. This must be done atomically to protect
against nested writers.

```
  temp_page = tail_page
  next_page = temp_page->next
  cmpxchg(tail_page, temp_page, next_page)
```

The above will update the tail page if it is still pointing to the expected
page. If this fails, a nested write pushed it forward, the the current write
does not need to push it.

```
           temp page
               |
               v
            tail page
               |
               v
    +---+    +---+    +---+    +---+
<---|   |--->|   |--->|   |--->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

Nested write comes in and moves the tail page forward:

```
                    tail page (moved by nested writer)
            temp page   |
               |        |
               v        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |--->|   |--->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

The above would fail the cmpxchg, but since the tail page has already
been moved forward, the writer will just try again to reserve storage
on the new tail page.

But the moving of the head page is a bit more complex.

```
            tail page
               |
               v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-H->|   |--->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

The write converts the head page pointer to UPDATE.

```
            tail page
```

```
               |
               v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-U->|   |--->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

But if a nested writer preempts here, it will see that the next
page is a head page, but it is also nested. It will detect that
it is nested and will save that information. The detection is the
fact that it sees the UPDATE flag instead of a HEADER or NORMAL
pointer.

The nested writer will set the new head page pointer.

```
           tail page
               |
               v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-U->|   |-H->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

But it will not reset the update back to normal. Only the writer
that converted a pointer from HEAD to UPDATE will convert it back
to NORMAL.

```
                    tail page
                        |
                        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-U->|   |-H->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

After the nested writer finishes, the outermost writer will convert
the UPDATE pointer to NORMAL.

```
                    tail page
                        |
                        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |--->|   |-H->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

It can be even more complex if several nested writes came in and moved
the tail page ahead several pages:

(first writer)

```
            tail page
               |
               v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-H->|   |--->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

The write converts the head page pointer to UPDATE.

```
            tail page
               |
               v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-U->|   |--->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

Next writer comes in, and sees the update and sets up the new
head page.

(second writer)

```
           tail page
               |
               v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-U->|   |-H->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

The nested writer moves the tail page forward. But does not set the old
update page to NORMAL because it is not the outermost writer.

```
                    tail page
                        |
                        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-U->|   |-H->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

Another writer preempts and sees the page after the tail page is a head page.
It changes it from HEAD to UPDATE.

(third writer)

```
                    tail page
                        |
                        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-U->|   |-U->|   |--->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

The writer will move the head page forward:

(third writer)

```
                    tail page
                        |
                        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-U->|   |-U->|   |-H->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

But now that the third writer did change the HEAD flag to UPDATE it
will convert it to normal:

(third writer)

```
                    tail page
                        |
                        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-U->|   |--->|   |-H->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

Then it will move the tail page, and return back to the second writer.

(second writer)

```
                             tail page
                                 |
                                 v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-U->|   |--->|   |-H->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

The second writer will fail to move the tail page because it was already
moved, so it will try again and add its data to the new tail page.
It will return to the first writer.

(first writer)

```
                             tail page
                                 |
                                 v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-U->|   |--->|   |-H->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

The first writer cannot know atomically if the tail page moved
while it updates the HEAD page. It will then update the head page to

what it thinks is the new head page.

```
(first writer)

                             tail page
                                 |
                                 v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-U->|   |-H->|   |-H->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

Since the cmpxchg returns the old value of the pointer the first writer
will see it succeeded in updating the pointer from NORMAL to HEAD.
But as we can see, this is not good enough. It must also check to see
if the tail page is either where it use to be or on the next page:

```
(first writer)

               A        B    tail page
               |        |        |
               v        v        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-U->|   |-H->|   |-H->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

If tail page != A and tail page != B, then it must reset the pointer
back to NORMAL. The fact that it only needs to worry about nested
writers means that it only needs to check this after setting the HEAD page.

```
(first writer)

               A        B    tail page
               |        |        |
               v        v        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |-U->|   |--->|   |-H->
--->|   |<---|   |<---|   |<---|   |<---
    +---+    +---+    +---+    +---+
```

Now the writer can update the head page. This is also why the head page must
remain in UPDATE and only reset by the outermost writer. This prevents
the reader from seeing the incorrect head page.

```
(first writer)

               A        B    tail page
               |        |        |
               v        v        v
    +---+    +---+    +---+    +---+
<---|   |--->|   |--->|   |--->|   |-H->
```

```
--->|    |<---|    |<---|    |<---|    |<---
    +----+    +----+    +----+    +----+
```