Read Copy Update

and the Linux Kernel

Houston PerlMongers
Thursday Sept 8th 2016
by Julian Brown

In our last episode

- We discussed various lock/wait free architectures
- These were:
  - CAS (Copy and Swap)
  - FAA (Fetch and Add)
  - We saw a Concurrent Linked List structure (using CAS)
  - And a Ticket Lock Queue system (using FAA)

Atomic Operators

- These mechanisms were implemented with primitive atomic operators, ensuring no corruption at certain points.

- Today we talk about another mechanism that is a “strategy” rather then an atomic operator, although it uses atomic operators.

Read Copy Update

- This discussion on RCU is straight from Wikipedia.
- In computer operating systems, read-copy-update (RCU) is a synchronization mechanism implementing a kind of mutual exclusion that can sometimes be used as an alternative to a readers-writer lock. It allows extremely low overhead, wait-free reads. However, RCU updates can be expensive, as they must leave the old versions of the data structure in place to accommodate pre-existing readers. These old versions are reclaimed after all pre-existing readers finish their accesses.
- The expense is not time, but space.
- I hate software patents, RCU was patented by Sequent Computer Systems (they were famous for multi-processing systems) and are now owned by IBM.

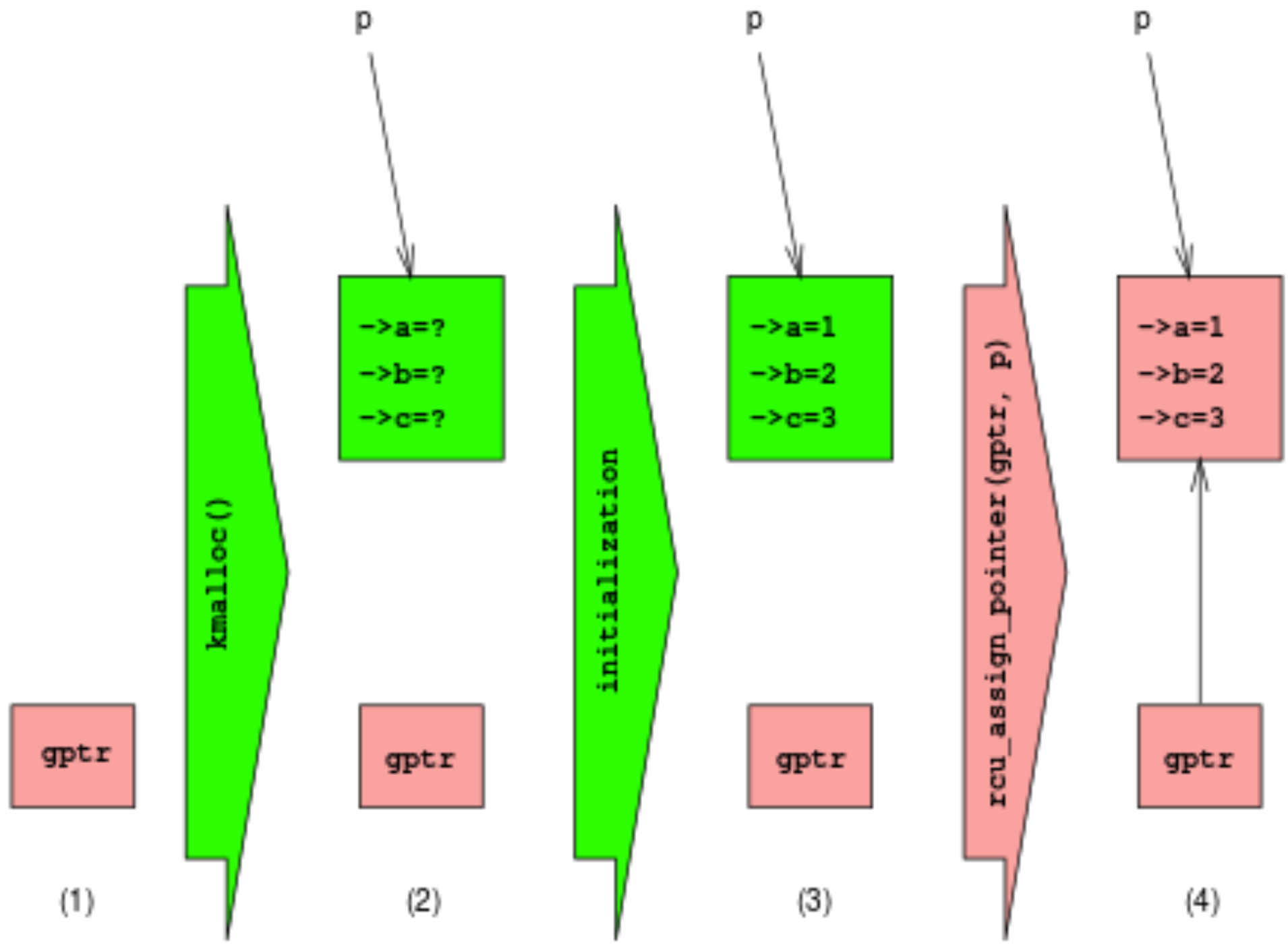
Properties

- Readers can access a data structure even when it is in the process of being updated.

- Updaters cannot block readers or force them to retry their accesses.

- Essentially, the updater leaves the existing structure in place for anyone currently using the structure. It updates the structure by copying and eventually garbage collecting the previous version when it is no longer being read.

Insertion


p
->a=?
->b=?
->c=?
gptr
kmalloc()
(1)
p
->a=?
->b=?
->c=?
gptr
(2)
p
->a=1
->b=2
->c=3
gptr
initialization
(3)
p
->a=1
->b=2
->c=3
gptr
rcu_assign_pointer(gptr, p)
(4)

Notes

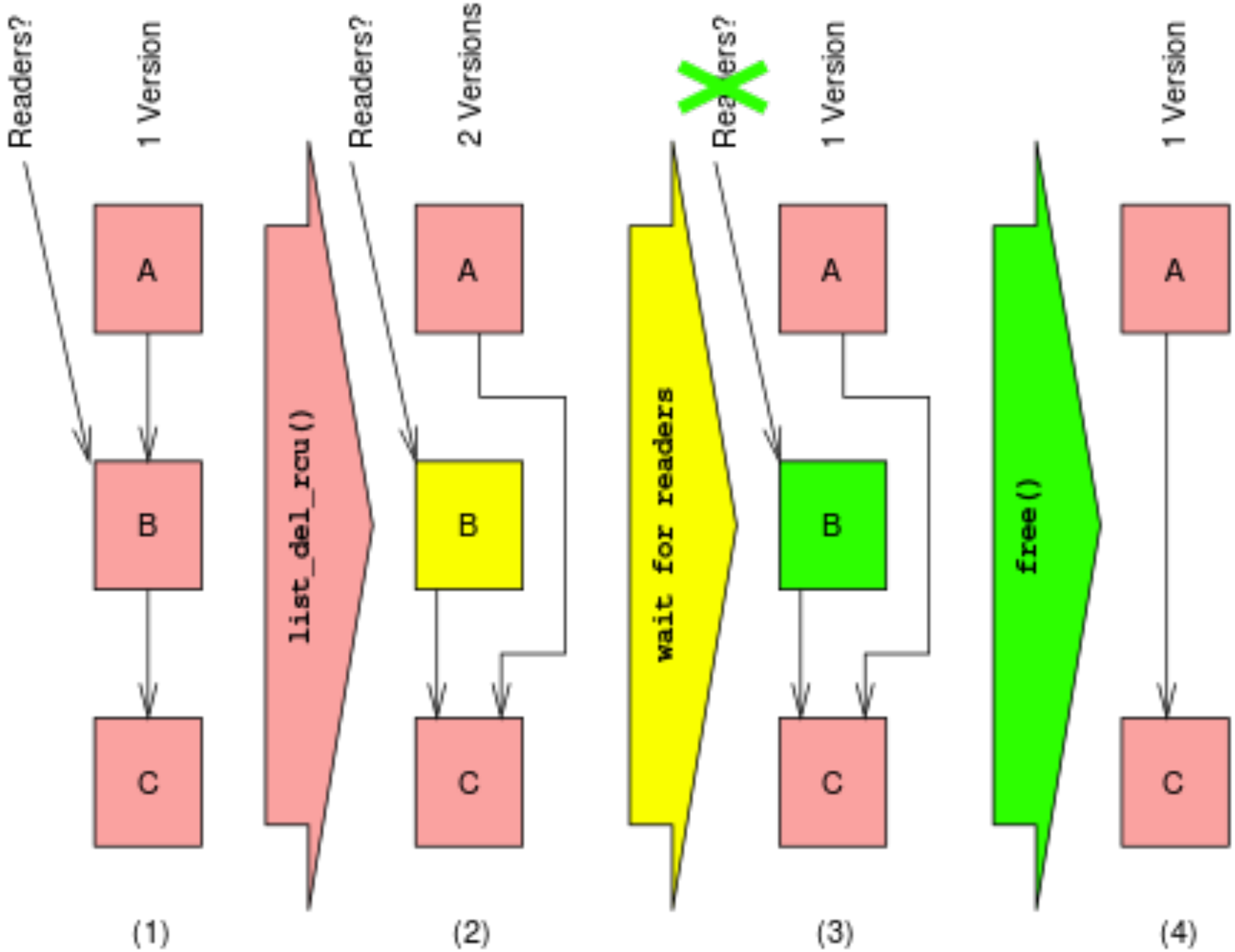
- “gptr” is pointing to an existing linked list

- “a” links to “b” links to “c”

Insertion Discussion

- Step 1
- “gptr” starts it’s life as NULL
- It is red to indicate it is could be read.
- Allocate memory for new copy
- Step 2
- Initialize Structure (a=1 …)
- Step 3
- rcu_assign_pointer (atomic, further discussion later)
- Step 4
- “gptr” is now up to date
- This procedure demonstrates how new data may be inserted into a linked data structure even though readers are concurrently traversing the data structure before, during, and after the insertion.
- This illustration seems weak and only used on initialization, the deletion really shows the good stuff.

Deletion


Readers?
1 Version
A
B
C
(1)
Readers?
2 Versions
A
list_del_rcu()
B
C
(2)
Readers?
1 Version
A
wait for readers
B
C
(3)
1 Version
A
free()
C
(4)

Delete Discussion

- The first state shows a linked list containing elements A, B, and C.
- All three elements are colored red to indicate that an RCU reader might reference any of them at any time.
- list_del_rcu removes element B from list. Leaving intact the link to C. This allows anyone still reading B to continue their traversal. This is atomic.
- Element B is yellow to show that while readers might still have a reference to B. New readers cannot see B.
- wait-for-readers operation need only wait for pre-existing readers not new readers.
- Now B is green and can be freed.

Notes

- RCU's readers execute within read-side critical sections, using rcu_read_lock and rcu_read_unlock. Not sure how that works yet.

- Any statement not in RCU reader lock is said to be quiescent and are not permitted to hold references to RCU pointers.

- Wait-for-readers does not wait for quiescent threads, only if there are current critical sections.

Linux Kernel

- The primary maintainer and champion of the RCU tools in the Kernel is Paul McKenney.

- One of the back-bone structures in the Kernel is the Radix Tree …

Brazenly stolen slides

http://events.linuxfoundation.jp/sites/events/files/slides/LinuxConNA2016%20-%20Radix%20Tree.pdf

What is a Radix Tree?

- Wikipedia says Radix Trees are all about strings.
  - Used for string compression and inverted indices of text documents
- Linux says Radix Trees are all about converting small integers to pointers
  - Julian's note, kinda like a hash in Perl (hey we all love Perl right?).
- The Linux Radix Tree appears to be an independent reinvention of the Judy Array.
  - Julian's note, no idea what a Judy Array is.
  - In Julian's opinion it looks like a BTree with a radix bucket at each node.

How does it work?

- Each node of the Radix tree contains 64 pointers.
  - The “next” 6 bits (2^6 = 64) of the index determine which pointer to use.
    - If this is the last level, the pointer is a user pointer.
    - If not the last level, the pointer points to the next node.
- Other tree metadata is also stored at each layer:
  - Tags, height (shift), reference count, parent pointer, offset in parent.

RCU and the Radix Tree

- With care, some radix tree functions can be used with only rcu_read_lock protection.
    - Which (depending on kernel config options) may mean no protection.
- Many CPU's may be walking the tree at the same time another CPU is inserting or deleting an entry from the tree.
- The user may get back a stale pointer from the tree walk, but it is guaranteed to be a pointer which was in the tree for that index at some point.
- Radix Tree frees tree nodes using RCU, so any CPU holding the read lock is guaranteed not to reference freed memory.

Back to less brazenly stolen slides :)

Lockless Page Cache

- The Page Cache has 2 parts:
  - The Cache itself, a pseudo Linked List
  - And a Radix Tree to manage the list.
- The Page Cache uses the Radix Tree as an index to the pages.
- The Radix Tree is already RCU.
- But to prevent corruption, it is still managed via a semaphore because of the 2 parts.
- W/O the semaphore a page could be evicted before it is accessed after getting the information on it via the Radix Tree.
- Nick Piggin came up with an RCU like change to make this lockless.
- int page_cache_get_speculative(struct page *page);
- If the given page has a reference count of zero, then the page has been removed and returns zero.
- If the reference count is non-zero it is incremented (FAA maybe?) and non zero will be returned.
- You try again if you get a zero return.
- When you are done you decrement the reference count and if zero it will be evicted from memory.

Attributions

- https://en.wikipedia.org/wiki/Read-copy-update
- http://kukuruku.co/hub/cpp/lock-free-data-structures-the-inside-rcu
- http://lwn.net/Articles/291826/
- http://lwn.net/Articles/275808/
- http://events.linuxfoundation.jp/sites/events/files/slides/LinuxConNA2016%20-%20Radix%20Tree.pdf