Read-Copy-Update (RCU)
Yet another kernel synchronization primitive

Vasileios P. Kemerlis

Network Security Lab
Computer Science Department
Columbia University
New York, NY

02/17/2010
Outline

1. Synchronization with Read-Copy-Update
   - Introduction
   - Usage

2. RCU Details
   - What is it anyway?
   - Update internals

3. RCU API
   - Core API
   - Current state
   - More reading
Outline

1. Synchronization with Read-Copy-Update
   - Introduction
   - Usage

2. RCU Details
   - What is it anyway?
   - Update internals

3. RCU API
   - Core API
   - Current state
   - More reading
RCU
Read-Copy-Update

- Kernel synchronization primitive (yet another)
- Added in 2.5 kernel branch
- Very popular in the kernel community
RCU
Why bother?

- Replacement for *reader/writer* locks
- Used in many places inside the kernel
- Performs very well (scalable, efficient, deterministic)
- In 2.6.31 more than 292 source files under `net/` utilize the RCU API
RCU
Why bother?

- Replacement for *reader/writer* locks
- Used in many places inside the kernel
- Performs very well (scalable, efficient, deterministic)
- In 2.6.31 more than 292 source files under `net/` utilize the RCU API
- **But** RCU has its cons also – *there is no silver bullet*
Outline

1. Synchronization with Read-Copy-Update
   - Introduction
   - Usage

2. RCU Details
   - What is it anyway?
   - Update internals

3. RCU API
   - Core API
   - Current state
   - More reading
1. **Synchronize** access to a protected resource
   - But access to the resource must be only via a pointer

2. No sleep inside an RCU region

3. It provides performance gains only if the resource is mostly read (i.e., sparse writers, <10% time spend in updating)
Outline

1. Synchronization with Read-Copy-Update
   - Introduction
   - Usage

2. RCU Details
   - What is it anyway?
   - Update internals

3. RCU API
   - Core API
   - Current state
   - More reading
RCU
In a nutshell

- It keeps track of all *pointers* that point to the shared resource
- When the resource is *modified*, a *copy* is first created and the change is performed on that copy
- After all *readers* are done with the previous (old) copy of the resource, their pointer is updated (now points to the new copy of the structure)
RCU
In a nutshell

- It keeps track of all *pointers* that point to the shared resource
- When the resource is *modified*, a *copy* is first created and the change is performed on that copy
- After all *readers* are done with the previous (old) copy of the resource, their pointer is updated (now points to the new copy of the structure)
- *More aggressive concurrency* – reads happen at the same time that a write is performed
Updates are split into removal and reclamation phases.

In the removal phase, all references to data items of the protected structure are replaced (now pointing to new versions).

During the reclamation phase, the old items are freed (garbage collection?)
Outline

1. Synchronization with Read-Copy-Update
   - Introduction
   - Usage

2. RCU Details
   - What is it anyway?
   - Update internals

3. RCU API
   - Core API
   - Current state
   - More reading
1. Remove the pointers of a data structure (i.e., subsequent readers cannot gain a reference to it)
RCU
Update internals

1. Remove the pointers of a data structure (i.e., subsequent readers cannot gain a reference to it)
2. Wait for all previous readers to complete their RCU read-side critical sections
1. Remove the pointers of a data structure (i.e., subsequent readers cannot gain a reference to it)
2. Wait for all previous readers to complete their RCU read-side critical sections
3. At this point, there cannot be any readers who hold references to the data structure, so it now may safely be reclaimed (e.g., kfree())
Outline

1 Synchronization with Read-Copy-Update
   - Introduction
   - Usage

2 RCU Details
   - What is it anyway?
   - Update internals

3 RCU API
   - Core API
   - Current state
   - More reading
RCU
Core API

- rcu_read_lock()
- rcu_read_unlock()
- synchronize_rcu()
- call_rcu()
- rcu_assign_pointer()
- rcu_dereference()
struct foo {
  int a;
  char b;
  long c;
};

DEFINE_SPINLOCK(foo_mutex);
struct foo *gbl_foo;
void foo_update_a(int new_a)
{
struct foo *new_fp;
struct foo *old_fp;

new_fp = kmalloc(sizeof(*new_fp), GFP_KERNEL);
spin_lock(&foo_mutex);
old_fp = gbl_foo;
*new_fp = *old_fp;
new_fp->a = new_a;
rcu_assign_pointer(gbl_foo, new_fp);
spin_unlock(&foo_mutex);
synchronize_rcu();
kfree(old_fp);
int foo_get_a(void)
{
    int retval;

    rcu_read_lock();
    retval = rcu_dereference(gbl_foo)->a;
    rcu_read_unlock();
    return retval;
}
RCU Example summary

- `rcu_read_lock()` and `rcu_read_unlock()` guard RCU read-side critical sections
- `rcu_dereference()` is used in order to dereference RCU-protected pointers
- Still need some solid scheme (e.g., spinlocks) to keep concurrent updates from interfering with each other
- `rcu_assign_pointer()` updates an RCU-protected pointer
- `synchronize_rcu()` is called after removing a data element from an RCU-protected data structure, but before reclaiming/freeing the data element
Outline

1. Synchronization with Read-Copy-Update
   - Introduction
   - Usage

2. RCU Details
   - What is it anyway?
   - Update internals

3. RCU API
   - Core API
   - Current state
   - More reading
Synchronization with Read-Copy-Update

RCU Details

RCU API

RCU
Evolution 0x1

The diagram shows the growth of the RCU API members over time, with the x-axis representing the years from 2002 to 2008 and the y-axis representing the number of RCU API members. The API members increased significantly from 2004 onwards, indicating a growing demand for its features and capabilities.
Synchronization with Read-Copy-Update

RCU Details

Core API

Current state

More reading

RCU
Evolution 0x2

![Graph showing the evolution of RCU API uses from 2002 to 2009. The x-axis represents the years from 2002 to 2009, and the y-axis represents the number of RCU API uses. The graph shows a steady increase in RCU API uses over the years.](image-url)
Outline

1. Synchronization with Read-Copy-Update
   - Introduction
   - Usage

2. RCU Details
   - What is it anyway?
   - Update internals

3. RCU API
   - Core API
   - Current state
   - More reading
The excellent LWN (http://lwn.net) “What is RCU?” series
1. “What is RCU, Fundamentally?”
   (http://lwn.net/Articles/262464/)
2. “What is RCU, Part 2: Usage”
   (http://lwn.net/Articles/263130/)
3. “What is RCU, RCU part 3: the RCU API”
   (http://lwn.net/Articles/264090/)

Paul McKenney’s papers
(http://www.rdrop.com/users/paulmck/RCU/)