Track Data Access in RCU Read-Side Critical Sections

Boqun Feng

Linux Technology Center, IBM

September 28, 2015

Track Data Access in RCU Read-Side Critical Sections

1. An Introduction to RCU
   - What is RCU?
   - RCU read-side critical sections
2. Why Tracking Data Accesses in RCU
3. Design and Implementation
   - Basics: lockdep
   - Design
   - Implementation

What is RCU?

a lock->rwlock->RCU case:

```
1   struct data {
2       int a;
3   };
4
5   struct data *gp = NULL;
6   spinlock_t lock;
7
8   void read()                                  void update(int a)
9   {                                            {
10      struct data *p;                              struct data *p;
11      spin_lock(&lock);                            spin_lock(&lock);
12      p = gp;                                      p = gp;
13      if (p)                                       if (!p) {
14          do_something_with(p->a);                     p = kmalloc(sizeof(*p));
15      spin_unlock(&lock);                              gp = p;
16  }                                                }
17                                                   p->a = a;
18                                                   spin_unlock(&lock);
19                                           }
```

Reader/Writer blocks reader/writer.

What is RCU? (cont.)

a lock->rwlock->RCU case:

```
1   struct data {
2       int a;
3   };
4
5   struct data *gp = NULL;
6   rwlock_t rwlock;
7
8   void read()                              void update(int a)
9   {                                        {
10      struct data *p;                          struct data *p;
11      spin_lock(&lock);                        write_lock(&lock);
12      p = gp;                                  p = gp;
13      if (p)                                   if (!p) {
14          do_something_with(p->a);                 p = kmalloc(sizeof(*p));
15      spin_unlock(&lock);                          gp = p;
16  }                                            }
17                                               p->a = a;
18                                               write_unlock(&lock);
19                                           }
```

Reader doesn't blocks reader.

What is RCU? (cont.)

a lock->rwlock->RCU case:

```
1   struct data {
2       int a;
3   };
4
5   spinlock_t update_lock;
6   struct data *gp = NULL;
7
8   void read()                                 void update(int a)
9   {                                           {
10      struct data *p;                             struct data *p, *old_p;
11      rcu_read_lock();                            p = kmalloc(sizeof(*p));
12      p = rcu_dereference(gp);                    p->a = a;
13      if (p)                                      spin_lock(&update_lock);
14          do_something_with(p->a);                old_p = gp;
15      rcu_read_unlock();                          rcu_assign_pointer(gp, p);
16  }                                               spin_unlock(&update_lock);
17                                                  synchronize_rcu();
18                                                  kfree(old_p);
19                                              }
```

Reader doesn't block reader/writer, and writer doesn't block reader.

RCU's fundamental guarantee

RCU guarantees allows writers to wait for all pre-existing RCU **read-side critical section**s to complete.

```
1
2     CPU 0 (reader)                  CPU 1 (writer)
3     ======================          ======================
4                                     ...
5                                     p = kmalloc(sizeof(*p));
6                                     p->a = a;
7     rcu_read_lock();                spin_lock(&update_lock);
8                                     old_p = gp;
9                                     rcu_assign_pointer(gp, p);
10                                    spin_unlock(&update_lock);
11    if (p)                          synchronize_rcu();
12        do_something_with(p->a);            |
13                                            |
14                                            |
15    rcu_read_unlock();                      |
16                                            V
17                                    kfree(old_p);
```

RCU read-side critical sections

An RCU read-side critical section

- begins with the marker **rcu_read_lock()**
- ends with the marker **rcu_read_unlock()**
- these markers may be nested

RCU treats a nested set as one big RCU read-side critical section.

```
1  rcu_read_lock(); <--------------------+
2  p1 = rcu_dereference(gp);             |
3      rcu_read_lock();                  |
4      p2 = rcu_dereference(gp);   on big RCU read-side critical section
5      rcu_read_unlock();                |
6  do_something_with(p2); // legal       |
7  rcu_read_unlock(); <------------------+
```

RCU read-side critical sections

An RCU read-side critical section

- begins with the marker **rcu_read_lock()**
- ends with the marker **rcu_read_unlock()**
- these markers may be nested

RCU treats a nested set as one big RCU read-side critical section.

```
1     rcu_read_lock(); <--------------------+
2     p1 = rcu_dereference(gp);               |
3         rcu_read_lock();                    |
4         p2 = rcu_dereference(gp);    on big RCU read-side critical section
5         rcu_read_unlock();                  |
6     do_something_with(p2); // legal         |
7     rcu_read_unlock(); <------------------+
```

Note: these markers don't have any parameter.

Track Data Access in RCU Read-Side Critical Sections

1. An Introduction to RCU
   - What is RCU?
   - RCU read-side critical sections
2. Why Tracking Data Accesses in RCU
3. Design and Implementation
   - Basics: lockdep
   - Design
   - Implementation

Data and critical sections

A lock used to belong to a data structure, but RCU?

```
1   struct dentry *dget_parent(struct dentry *dentry)
2   {
3       struct dentry *ret;
4       ...
5       repeat:
6           rcu_read_lock(); <---------------------------+
7           ret = dentry->d_parent;        what is this critical section protecting?
8           spin_lock(&ret->d_lock); <-----------------+  |
9           if (unlikely(ret != dentry->d_parent)) { the critical section to protect @ret
10              spin_unlock(&ret->d_lock); <----------+  |
11              rcu_read_unlock(); <--------------------+
12          }
13          rcu_read_unlock();
14          sin_unlock(&ret->d_lock);
15          goto repeat;
16
17  }
```

Torward a data oriented RCU

- Ingo Molnar:
  http://thread.gmane.org/gmane.linux.kernel/1930157/focus=1930821

Torward a data oriented RCU

- Ingo Molnar: http://thread.gmane.org/gmane.linux.kernel/1930157/focus=1930821
  - add a parameter for rcu_read_lock() and rcu_read_unlock().

Torward a data oriented RCU

- Ingo Molnar: http://thread.gmane.org/gmane.linux.kernel/1930157/focus=1930821
  - add a parameter for rcu_read_lock() and rcu_read_unlock().
  - But there are more than 1000 callsites of these primitives

Torward a data oriented RCU

- Ingo Molnar: http://thread.gmane.org/gmane.linux.kernel/1930157/focus=1930821
  - add a parameter for rcu_read_lock() and rcu_read_unlock().
  - But there are more than 1000 callsites of these primitives
  - And it's normal to put two RCU-protected data into one critical section.

Torward a data oriented RCU

- Ingo Molnar:
  http://thread.gmane.org/gmane.linux.kernel/1930157/focus=1930821
  - add a parameter for rcu_read_lock() and rcu_read_unlock().
  - But there are more than 1000 callsites of these primitives
  - And it's normal to put two RCU-protected data into one critical section.
- Paul Mckenney:

Torward a data oriented RCU

- Ingo Molnar: http://thread.gmane.org/gmane.linux.kernel/1930157/focus=1930821
  - add a parameter for rcu_read_lock() and rcu_read_unlock().
  - But there are more than 1000 callsites of these primitives
  - And it's normal to put two RCU-protected data into one critical section.
- Paul Mckenney:
  - if we could do a lockdep-like association of the rcu_dereference() calls with the rcu_read_lock() calls that protected them

Torward a data oriented RCU

- Ingo Molnar: http://thread.gmane.org/gmane.linux.kernel/1930157/focus=1930821
  - add a parameter for rcu_read_lock() and rcu_read_unlock().
  - But there are more than 1000 callsites of these primitives
  - And it's normal to put two RCU-protected data into one critical section.
- Paul Mckenney:
  - if we could do a lockdep-like association of the rcu_dereference() calls with the rcu_read_lock() calls that protected them
  - and used lockdep to associate rcu_assign_pointer() with the locks protecting it.

Torward a data oriented RCU

- Ingo Molnar: http://thread.gmane.org/gmane.linux.kernel/1930157/focus=1930821
  - add a parameter for rcu_read_lock() and rcu_read_unlock().
  - But there are more than 1000 callsites of these primitives
  - And it's normal to put two RCU-protected data into one critical section.
- Paul Mckenney:
  - if we could do a lockdep-like association of the rcu_dereference() calls with the rcu_read_lock() calls that protected them
  - and used lockdep to associate rcu_assign_pointer() with the locks protecting it.
  - We could easily find the set of rcu_dereference() calls accessing that same RCU-protected pointer,

Torward a data oriented RCU

- Ingo Molnar: http://thread.gmane.org/gmane.linux.kernel/1930157/focus=1930821
  - add a parameter for rcu_read_lock() and rcu_read_unlock().
  - But there are more than 1000 callsites of these primitives
  - And it's normal to put two RCU-protected data into one critical section.
- Paul Mckenney:
  - if we could do a lockdep-like association of the rcu_dereference() calls with the rcu_read_lock() calls that protected them
  - and used lockdep to associate rcu_assign_pointer() with the locks protecting it.
  - We could easily find the set of rcu_dereference() calls accessing that same RCU-protected pointer,
  - and then associate the update-side locks with the RCU read-side critical sections.

Torward a data oriented RCU

- Ingo Molnar: http://thread.gmane.org/gmane.linux.kernel/1930157/focus=1930821
  - add a parameter for rcu_read_lock() and rcu_read_unlock().
  - But there are more than 1000 callsites of these primitives
  - And it's normal to put two RCU-protected data into one critical section.
- Paul Mckenney:
  - if we could do a lockdep-like association of the rcu_dereference() calls with the rcu_read_lock() calls that protected them
  - and used lockdep to associate rcu_assign_pointer() with the locks protecting it.
  - We could easily find the set of rcu_dereference() calls accessing that same RCU-protected pointer,
  - and then associate the update-side locks with the RCU read-side critical sections.

Torward a data oriented RCU

- Ingo Molnar: http://thread.gmane.org/gmane.linux.kernel/1930157/focus=1930821
  - add a parameter for rcu_read_lock() and rcu_read_unlock().
  - But there are more than 1000 callsites of these primitives
  - And it's normal to put two RCU-protected data into one critical section.
- Paul Mckenney:
  - if we could do a lockdep-like association of the rcu_dereference() calls with the rcu_read_lock() calls that protected them
  - and used lockdep to associate rcu_assign_pointer() with the locks protecting it.
  - We could easily find the set of rcu_dereference() calls accessing that same RCU-protected pointer,
  - and then associate the update-side locks with the RCU read-side critical sections.

And here comes the lockdep-like association of step 1.

Track Data Access in RCU Read-Side Critical Sections

1. An Introduction to RCU
   - What is RCU?
   - RCU read-side critical sections
2. Why Tracking Data Accesses in RCU
3. Design and Implementation
   - Basics: lockdep
   - Design
   - Implementation

Lockdep

Lockdep is the runtime locking correctness validator in kernel. Concepts:

- held_lock: describe a task is holding some locks, pointers to a lockdep_map.

Basic Idea:

Suit for

Lockdep

Lockdep is the runtime locking correctness validator in kernel. Concepts:

- held_lock: describe a task is holding some locks, pointers to a lockdep_map.
- lockdep_map: one per lock instance, presenting the instance in lockdep framework, pointers to a lock_class.

Basic Idea:

Suit for

Lockdep

Lockdep is the runtime locking correctness validator in kernel. Concepts:

- held_lock: describe a task is holding some locks, pointers to a lockdep_map.
- lockdep_map: one per lock instance, presenting the instance in lockdep framework, pointers to a lock_class.
- lock_class: a group of lock instances, grouped by users.

Basic Idea:

Suit for

Lockdep

Lockdep is the runtime locking correctness validator in kernel. Concepts:

- held_lock: describe a task is holding some locks, pointers to a lockdep_map.
- lockdep_map: one per lock instance, presenting the instance in lockdep framework, pointers to a lock_class.
- lock_class: a group of lock instances, grouped by users.

Basic Idea:

- Track the 'state' of lock-classes and dependencies between different lock-classes.

Suit for

Lockdep

Lockdep is the runtime locking correctness validator in kernel. Concepts:

- held_lock: describe a task is holding some locks, pointers to a lockdep_map.
- lockdep_map: one per lock instance, presenting the instance in lockdep framework, pointers to a lock_class.
- lock_class: a group of lock instances, grouped by users.

Basic Idea:

- Track the 'state' of lock-classes and dependencies between different lock-classes.
- Prove that the state and the dependencies are correct.

Suit for

Lockdep

Lockdep is the runtime locking correctness validator in kernel. Concepts:

- held_lock: describe a task is holding some locks, pointers to a lockdep_map.
- lockdep_map: one per lock instance, presenting the instance in lockdep framework, pointers to a lock_class.
- lock_class: a group of lock instances, grouped by users.

Basic Idea:

- Track the 'state' of lock-classes and dependencies between different lock-classes.
- Prove that the state and the dependencies are correct.

Suit for

- Finding dead locks

Lockdep

Lockdep is the runtime locking correctness validator in kernel. Concepts:

- held_lock: describe a task is holding some locks, pointers to a lockdep_map.
- lockdep_map: one per lock instance, presenting the instance in lockdep framework, pointers to a lock_class.
- lock_class: a group of lock instances, grouped by users.

Basic Idea:

- Track the 'state' of lock-classes and dependencies between different lock-classes.
- Prove that the state and the dependencies are correct.

Suit for

- Finding dead locks
- Recording stacks/chains of critical sections

Lockdep

Lockdep is the runtime locking correctness validator in kernel. Concepts:

- held_lock: describe a task is holding some locks, pointers to a lockdep_map.
- lockdep_map: one per lock instance, presenting the instance in lockdep framework, pointers to a lock_class.
- lock_class: a group of lock instances, grouped by users.

Basic Idea:

- Track the 'state' of lock-classes and dependencies between different lock-classes.
- Prove that the state and the dependencies are correct.

Suit for

- Finding dead locks
- Recording stacks/chains of critical sections
- Accounting usage of critical sections.

Lockdep for RCU

All RCU read critical sections of each flavor share a same lock class.

- For viewpoint of lockdep, RCU is the same as global read locks.

Lockdep for RCU

All RCU read critical sections of each flavor share a same lock class.

- For viewpoint of lockdep, RCU is the same as global read locks.

IOW, lockdep can't build relation between data and RCU read-side critical sections.

Design

Principles:

- Don’t change the API of RCU

Basic Ideas:

Design

Principles:

- Don’t change the API of RCU
- Reuse lockdep framework as much as possible.

Basic Ideas:

Design

Principles:

- Don’t change the API of RCU
- Reuse lockdep framework as much as possible.
- Take potential users(other than RCU) into consideration.

Basic Ideas:

Design

Principles:

- Don't change the API of RCU
- Reuse lockdep framework as much as possible.
- Take potential users(other than RCU) into consideration.

Basic Ideas:

- An RCU lock+unlock pair(RCU context) can be identified by the callsite of rcu_read_lock().

Design

Principles:

- Don't change the API of RCU
- Reuse lockdep framework as much as possible.
- Take potential users(other than RCU) into consideration.

Basic Ideas:

- An RCU lock+unlock pair(RCU context) can be identified by the callsite of rcu_read_lock().
- Therefore, we can track the stack of current RCU contexts.

Design

Principles:

- Don't change the API of RCU
- Reuse lockdep framework as much as possible.
- Take potential users(other than RCU) into consideration.

Basic Ideas:

- An RCU lock+unlock pair(RCU context) can be identified by the callsite of rcu_read_lock().
- Therefore, we can track the stack of current RCU contexts.
  - Lockdep has done partial work.

Design

Principles:

- Don't change the API of RCU
- Reuse lockdep framework as much as possible.
- Take potential users(other than RCU) into consideration.

Basic Ideas:

- An RCU lock+unlock pair(RCU context) can be identified by the callsite of rcu_read_lock().
- Therefore, we can track the stack of current RCU contexts.
  - Lockdep has done partial work.
- Based on which, we can know which data is accessed in which RCU context.

Design

Principles:

- Don't change the API of RCU
- Reuse lockdep framework as much as possible.
- Take potential users(other than RCU) into consideration.

Basic Ideas:

- An RCU lock+unlock pair(RCU context) can be identified by the callsite of rcu_read_lock().
- Therefore, we can track the stack of current RCU contexts.
  - Lockdep has done partial work.
- Based on which, we can know which data is accessed in which RCU context.
  - The datas used for tracking are all static generated.

Implementation overview

- Static variables to indicate RCU contexts and data accesses
- Callbacks in lock instance to reuse the lockdep framework

Static variables indicating RCU contexts and data accesses

Introduce struct lock_location and rcu_context:

```
1   struct lock_location {
2       char *filename;
3       long lineno;
4   };
5   #define LOCK_LOCATION(type) { .filename = __FILE__, .lineno = __LINE__ }
6
7   struct rcu_context {
8       struct lock_location *begin;
9       struct lock_location *end;
10      ...
11  };
```

Static variables indicating RCU contexts and data accesses(cont.)

- in rcu_read_lock() (switch back to macro):

```
1               {
2               ...
3               static struct lock_location begin = LOCK_LOCATION(); \
4               static struct rcu_context context = { .begin = &begin }; \
5               lock_acquire(map, 0, 0, 2, 0, NULL, (unsigned long)&context); \
6               ...
7               }
```

- in rcu_dereference():

```
1               {
2               ...
3               static struct lock_location loc = LOCK_LOCATION(); \
4               lock_access(&loc, ...);
5               ...
6               }
```

Callbacks in struct lock_map to reuse the lockdep framework

Callback interfaces:

```
1   struct lock_access_ops {
2       // called when a lock instance is acquired.
3       int (*acquire)(struct lockdep_map *lock,
4                      int subclass,
5                      unsigned long ip);
6       // called when a lock instance is released.
7       int (*release)(struct lockdep_map *lock,
8                      int subclass,
9                      unsigned long ip);
10      // called when a data is accessed in the critical sections of the lock instance.
11      int (*access)(struct lockdep_map *lock,
12                    struct held_lock *hlock,
13                    struct lock_location *loc,
14                    int type);
15  };
```

Register into lock instances

```
1   struct lockdep_map {
2       ...
3       struct lock_access_ops *ops;
4   };
```

Callbacks in struct lock_map to reuse the lockdep framework(cont.)

```
1   void lock_access(...)
2   {
3       ...
4       for (i = depth - 1; i >= 0; i--) {
5           hlock = curr->held_locks + i;
6           lock = hlock->instance;
7           if (lock->ops) {
8               lock->ops->access(lock->ops, hlock, loc, type);
9           }
10      }
11      ...
12  }
```

Result

Code mentioned by Ingo:

```
static int validate_change(struct cpuset *cur, struct cpuset *trial)
{
    ...
    rcu_read_lock();

    if (is_cpu_exclusive(cur) &&
        !cpuset_cpumask_can_shrink(cur->cpus_allowed,
                                   trial->cpus_allowed))
        goto out;

    ret = 0;
out:
    rcu_read_unlock();
    return ret;
}
```

cat /proc/rcu_lock_access

```
...
dereference at kernel/sched/core.c 2256:
rcu_context: kernel/sched/core.c 5525
rcu_context: kernel/cpuset.c 459
...
```

Conclusion and Future Work

Conclusion:

- RCU read-side critical sections are now data oriented.

Conclusion and Future Work

Conclusion:

- RCU read-side critical sections are now data oriented.
- Lockdep is a good start for recording accesses in critical sections.

Conclusion and Future Work

Conclusion:

- RCU read-side critical sections are now data oriented.
- Lockdep is a good start for recording accesses in critical sections.

Conclusion and Future Work

Conclusion:

- RCU read-side critical sections are now data oriented.
- Lockdep is a good start for recording accesses in critical sections.

Future Work:

- associate rcu_assign_pointer() with locks.
- associate rcu_assign_pointer() with rcu_dereference(). (Maybe hard but interesting)

Q & A

Thank you!