Linux RCU 分析

涂浩新

2018.3.29

RCU问答对

- 1.什么是RCU?
- 2.RCU的最佳应用场景是什么？
- 3.在哪里能找到更多关于Linux RCU的相关资料？
- 4.RCU的实现过程是怎么样的？
- 5.RCU在Linux kernel中覆盖的范围如何？

1.什么是RCU?

- RCU（Read-Copy Update）是数据同步的一种方式，在当前的Linux内核中发挥着重要的作用。
- RCU主要针对的数据对象是链表，目的是提高遍历读取数据的效率，为了达到目的使用RCU机制读取数据的时候不对链表进行耗时的加锁操作。这样在同一时间可以有多个线程同时读取该链表，并且允许一个线程对链表进行修改（修改的时候，需要加锁）。

2.RCU的最佳应用场景是什么？

- RCU适用于需要频繁的读取数据，而相应修改数据并不多的情景，例如在文件系统中，经常需要查找定位目录，而对目录的修改相对来说并不多，这就是RCU发挥作用的最佳场景。

3.在哪里能找到更多关于Linux RCU的相关资料？

- Linux内核源码当中,关于RCU的文档比较齐全，你可以在 /Documentation/RCU/ 目录下找到这些文件。
- Paul E. McKenney 是内核中RCU源码的主要实现者，他也写了很多RCU方面的文章。他把这些文章和一些关于RCU的论文的链接整理到了一起。相应链接如下：

  http://www2.rdrop.com/users/paulmck/RCU/

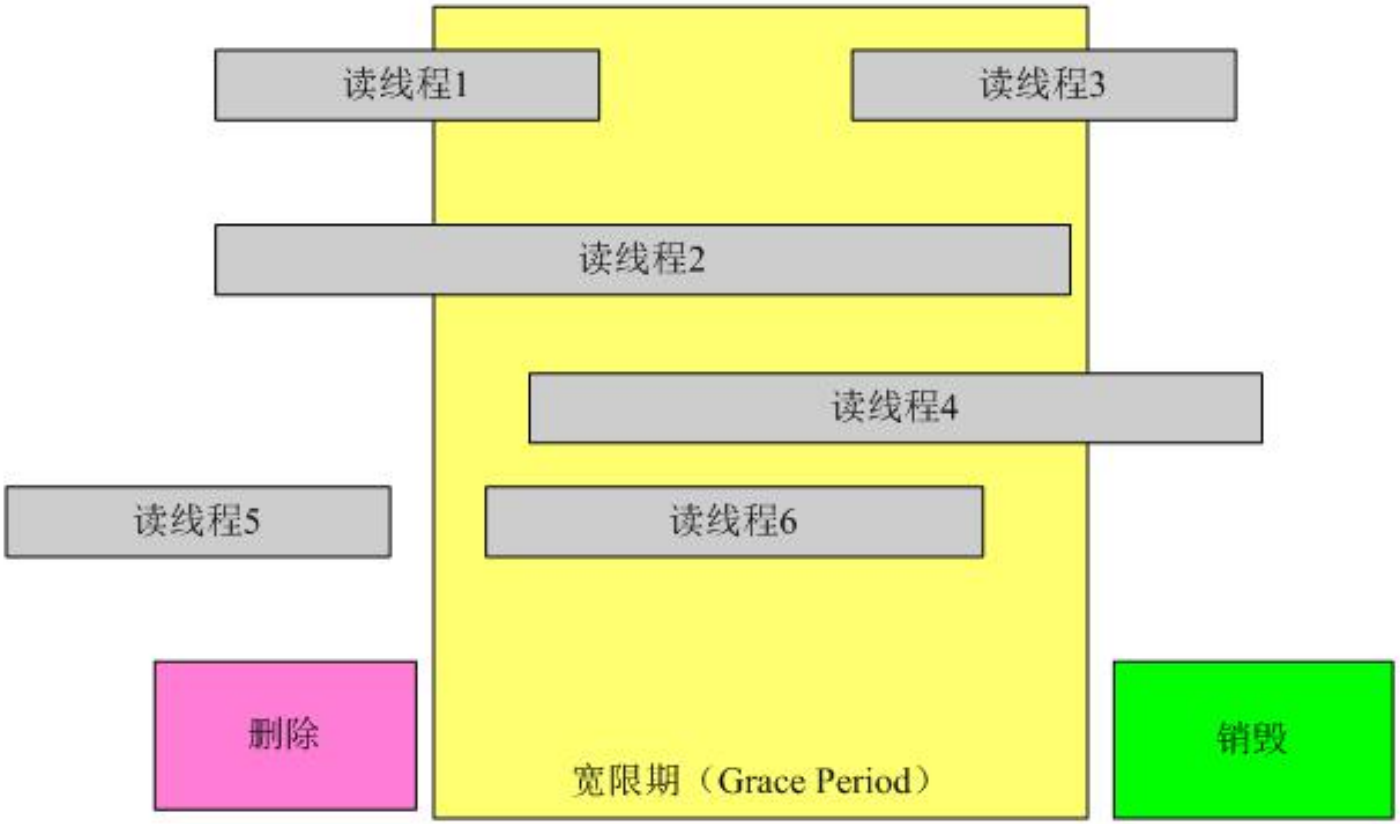
4.RCU的实现过程是怎么样的？

- 问题一：在读取过程中，另外一个线程删除了一个节点。删除线程可以把这个节点从链表中移除，但它不能直接销毁这个节点，必须等到所有的读取线程读取完成以后，才进行销毁操作。RCU中把这个过程称为宽限期（Grace period）。
- 问题二：在读取过程中，另外一个线程插入了一个新节点，而读线程读到了这个节点，那么需要保证读到的这个节点是完整的。这里涉及到了发布-订阅机制（Publish-Subscribe Mechanism）。
- 问题三：保证读取链表的完整性。新增或者删除一个节点，不至于导致遍历一个链表从中间断开。但是RCU并不保证一定能读到新增的节点或者不读到要被删除的节点。

解决问题一：宽限期

```
1.  struct foo {
2.            int a;
3.            char b;
4.            long c;
5.   };
6.
7.  DEFINE_SPINLOCK(foo_mutex);
8.
9.  struct foo *gbl_foo;
10.
11. void foo_read (void)
12. {
13.      foo *fp = gbl_foo;
14.      if ( fp != NULL )
15.             dosomething(fp->a, fp->b , fp->c );
16. }
17.
18. void foo_update( foo* new_fp )
19. {
20.      spin_lock(&foo_mutex);
21.      foo *old_fp = gbl_foo;
22.      gbl_foo = new_fp;
23.      spin_unlock(&foo_mutex);
24.      kfee(old_fp);
25. }
```

解决问题一：宽限期



解决问题一：宽限期

```
1.  void foo_read(void)
2.  {
3.      rcu_read_lock();
4.      foo *fp = gbl_foo;
5.      if ( fp != NULL )
6.              dosomething(fp->a,fp->b,fp->c);
7.      rcu_read_unlock();
8.  }
9.
10. void foo_update( foo* new_fp )
11. {
12.     spin_lock(&foo_mutex);
13.     foo *old_fp = gbl_foo;
14.     gbl_foo = new_fp;
15.     spin_unlock(&foo_mutex);
16.     synchronize_rcu();
17.     kfee(old_fp);
18. }
```

解决问题二：订阅-发布机制

```
1.  void foo_update( foo* new_fp )
2.  {
3.      spin_lock(&foo_mutex);
4.      foo *old_fp = gbl_foo;
5.
6.      new_fp->a = 1;
7.      new_fp->b = 'b';
8.      new_fp->c = 100;
9.
10.     gbl_foo = new_fp;
11.     spin_unlock(&foo_mutex);
12.     synchronize_rcu();
13.     kfee(old_fp);
14. }
```

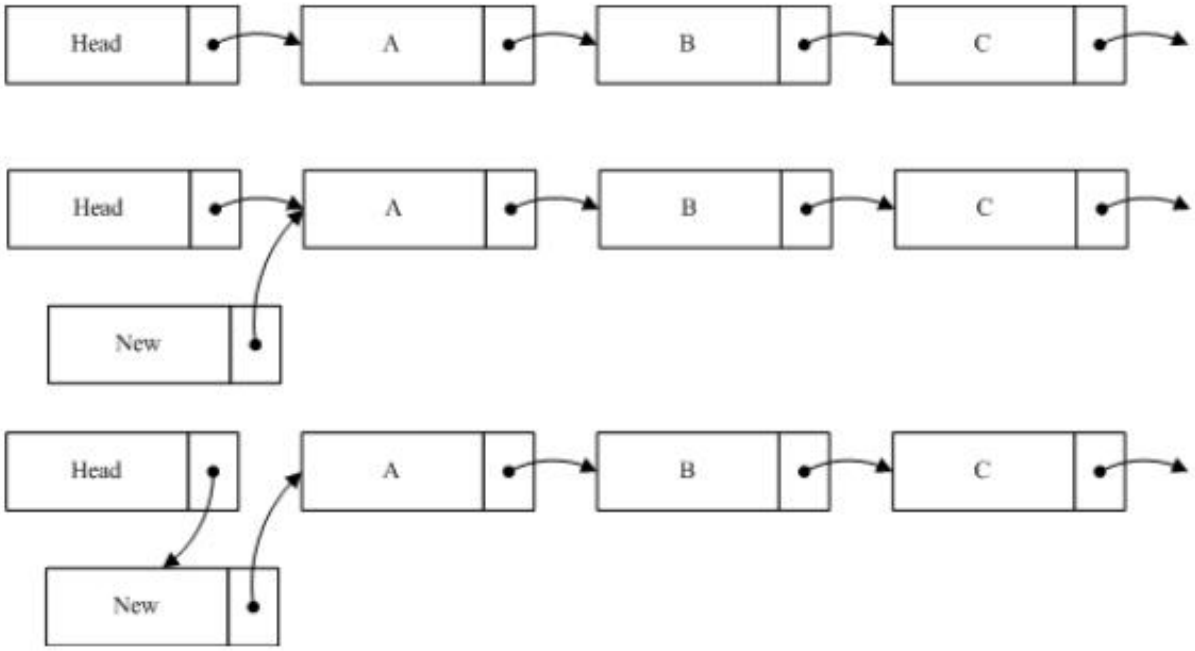
解决问题二：订阅-发布机制

- 将第十行修改为
  rcu_assign_pointer(gbl_foo,new_fp);

```
1.  #define rcu_assign_pointer(p, v) \
2.          __rcu_assign_pointer((p), (v), __rcu)
3.
4.  #define __rcu_assign_pointer(p, v, space) \
5.          do { \
6.                  smp_wmb(); \
7.                  (p) = (typeof(*v) __force space *)(v); \
8.          } while (0)
```

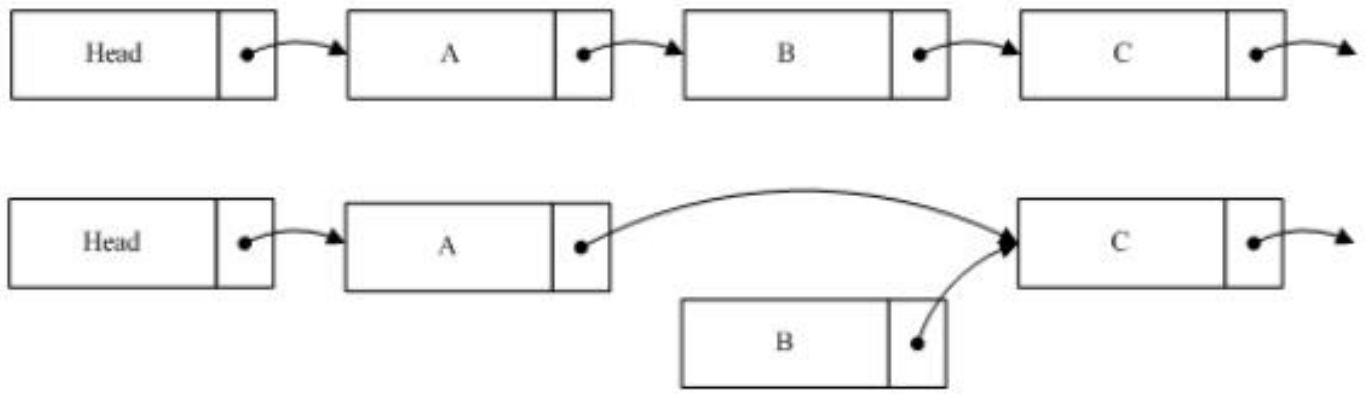
解决问题三：数据读取的完整性

- 增加一个节点



解决问题三：数据读取的完整性

- 删除一个节点



5.RCU在Linux kernel中覆盖的范围如何？

- Linux 3.7 RCU usage by subsystem

Subsystem	Uses	LoC	Uses / KLoC
virt	72	6,749	10.67
net	3251	740,382	4.39
ipc	35	8,306	4.21
security	251	68,494	3.66
kernel	628	198,304	3.17
mm	196	88,904	2.20
block	58	27,975	2.07
lib	70	52,235	1.34
fs	666	1,057,713	0.63
init	2	3,382	0.59
include	279	552,507	0.50
crypto	12	64,537	0.19
drivers	1061	8,530,160	0.12
arch	183	2,459,105	0.07
Total	6764	13,858,753	0.49

5.RCU在Linux kernel中覆盖的范围如何？

- Linux 3.7 RCU usage by RCU API function

Type of Usage	API Usage
RCU critical sections	3035
RCU dereference	972
RCU synchronization	696
RCU list traversal	574
RCU list update	524
RCU assign	358
Annotation of RCU-protected pointers	304
Initialization and cleanup	273
RCU lockdep assertion	28
Total	6764

谢谢！

涂浩新

2018.3.29