RCU机制是Linux2.6之后提供的一种数据一致性访问的机制，从RCU（read-copy-update）的名称上看，我们就能对他的实现机制有一个大概的了解，在修改数据的时候，首先需要读取数据，然后生成一个副本，对副本进行修改，修改完成之后再将老数据update成新的数据，此所谓RCU。

在操作系统中，数据一致性访问是一个非常重要的部分，通常我们可以采用锁机制实现数据的一致性访问。例如，semaphore、spinlock机制，在访问共享数据时，首先访问锁资源，在获取锁资源的前提下才能实现数据的访问。这种原理很简单，根本的思想就是在访问临界资源时，首先访问一个全局的变量（锁），通过全局变量的状态来控制线程对临界资源的访问。但是，这种思想是需要硬件支持的，硬件需要配合实现全局变量（锁）的读-修改-写，现代CPU都会提供这样的原子化指令。采用锁机制实现数据访问的一致性存在如下两个问题：

1、效率问题。锁机制的实现需要对内存的原子化访问，这种访问操作会破坏流水线操作，降低了流水线效率。这是影响性能的一个因素。另外，在采用读写锁机制的情况下，写锁是排他锁，无法实现写锁与读锁的并发操作，在某些应用下回降低性能。

2、扩展性问题。当系统中CPU数量增多的时候，采用锁机制实现数据的同步访问效率偏低。并且随着CPU数量的增多，效率降低，由此可见锁机制实现的数据一致性访问扩展性差。

为了解决上述问题，Linux中引进了RCU机制。该机制在多CPU的平台上比较适用，对于读多写少的应用尤其适用。RCU的思路实际上很简单。

1、对于读操作，可以直接对共享资源进行访问，但是前提是需要CPU支持访存操作的原子化，现代CPU对这一点都做了保证。但是RCU的读操作上下文是不可抢占的（这一点在下面解释），所以读访问共享资源时可以采用read_rcu_lock()，该函数的工作是停止抢占。

2、对于写操作，其需要将原来的老数据作一次备份（copy），然后对备份数据进行修改，修改完毕之后再用新数据更新老数据，更新老数据时采用了rcu_assign_pointer（ ）宏，在该函数中首先屏障一下memory，然后修改老数据。这个操作完成之后，需要进行老数据资源的回收。操作线程向系统注册回收方法，等待回收。采用数据备份的方法可以实现读者与写者之间的并发操作，但是不能解决多个写着之间的同步，所以当存在多个写者时，需要通过锁机制对其进行互斥，也就是在同一时刻只能存在一个写者。

3、在RCU机制中存在一个垃圾回收的daemon，当共享资源被update之后，可以采用该daemon实现老数据资源的回收。回收时间点就是在update之前的所有的读者全部退出。由此可见写者在update之后是需要睡眠等待的，需要等待读者完成操作，如果在这个时刻读者被抢占或者睡眠，那么很可能会导致系统死锁。因为此时写者在等待读者，读者被抢占或者睡眠，如果正在运行的线程需要访问读者和写者已经占用的资源，那么死锁的条件就很有可能形成了。

从上述分析来看，RCU思想是比较简单的，其核心内容紧紧围绕“写时拷贝”，采用RCU机制，能够保证在读写操作共享资源时，基本不需要取锁操作，能够在一定程度上提升性能。但是该机制的应用是有条件的，对于读多写少的应用，机制的开销比较小，性能会大幅度提升，但是如果写操作较多时，开销将会增大，性能不一定会有所提升。总体来说，RCU机制是对rw_lock的一种优化。