

## [PATCH] keys: Discard key spinlock and use RCU for key payload

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-03/3227.html>

---

*From:* David Howells ([dhowells\\_at\\_redhat.com](mailto:dhowells_at_redhat.com))

*Date:* 03/09/05

To: [torvalds@osdl.org](mailto:torvalds@osdl.org), [akpm@osdl.org](mailto:akpm@osdl.org), [trond.myklebust@fys.uio.no](mailto:trond.myklebust@fys.uio.no)

Date: Wed, 09 Mar 2005 17:59:15 +0000

The attached patch changes the key implementation in a number of ways:

- (1) It removes the spinlock from the key structure.
- (2) The key flags are now accessed using atomic bitops instead of write-locking the key spinlock and using C bitwise operators.

The three instantiation flags are dealt with with the construction semaphore held during the request\_key/instantiate/negate sequence, thus rendering the spinlock superfluous.

The key flags are also now bit numbers not bit masks.

- (3) The key payload is now accessed using RCU. This permits the recursive keyring search algorithm to be simplified greatly since no locks need be taken other than the usual RCU preemption disablement. Searching now does not require any locks or semaphores to be held; merely that the starting keyring be pinned.
- (4) The keyring payload now includes an RCU head so that it can be disposed of by call\_rcu(). This requires that the payload be copied on unlink to prevent introducing races in copy-down vs search-up.
- (5) The user key payload is now a structure with the data following it. It includes an RCU head like the keyring payload and for the same reason. It also contains a data length because the data length in the key may be changed on another CPU whilst an RCU protected read is in progress on the payload. This would then see the supposed RCU payload and the on-key data length getting out of sync.

I'm tempted to drop the key's datalen entirely, except that it's used in conjunction with quota management and so is a little tricky to get rid of.

Signed-Off-By: David Howells <[dhowells@redhat.com](mailto:dhowells@redhat.com)>

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

---

```
warthog>diffstat -p1 keys-rcu-2611.diff
```

```
Documentation/keys.txt      |    5
include/linux/key-ui.h     |    6 -
include/linux/key.h        |   25 ++--
security/keys/key.c        |   95 ++++++-----
security/keys/keyctl.c     |   23 +--
security/keys/keyring.c    |  250 ++++++-----
security/keys/proc.c       |   21 +-
security/keys/process_keys.c |   16 --
security/keys/request_key.c |   32 +----
security/keys/user_defined.c |   87 ++++++-----
```

10 files changed, 316 insertions(+), 244 deletions(-)

```
diff -uNr linux-2.6.11/Documentation/keys.txt linux-2.6.11-keys-rcu/Documentation/keys.txt
--- linux-2.6.11/Documentation/keys.txt 2005-01-04 11:12:42.000000000 +0000
+++ linux-2.6.11-keys-rcu/Documentation/keys.txt 2005-03-09 16:28:50.000000000 +0000
@@ -606,8 +606,9 @@
 due to two different users opening the same file is left to the filesystem
 author to solve.
```

-When accessing a key's payload data, the key->lock should be at least read-locked, or else the data may be changed by update during the access.  
+When accessing a key's payload data, RCU reading precautions on the payload pointer should be taken, or else the data may be changed by update during the access.

(\*) To search for a key, call:

```
diff -uNr linux-2.6.11/include/linux/key.h linux-2.6.11-keys-rcu/include/linux/key.h
--- linux-2.6.11/include/linux/key.h 2005-01-04 11:13:54.000000000 +0000
+++ linux-2.6.11-keys-rcu/include/linux/key.h 2005-03-09 16:48:34.284560505 +0000
@@ -18,7 +18,7 @@
```

```
#include <linux/types.h>
#include <linux/list.h>
#include <linux/rbtree.h>
-#include <linux/spinlock.h>
+#include <linux/rcupdate.h>
#include <asm/atomic.h>
```

```
#ifdef __KERNEL__
```

```
@@ -77,7 +77,6 @@
```

```
key_serial_t      serial;          /* key serial number */
struct rb_node    serial_node;
struct key_type   *type;          /* type of key */
- rwlock_t        lock;          /* examination vs change lock */
struct rw_semaphore sem;         /* change vs change sem */
struct key_user   *user;         /* owner of this key */
time_t           expiry;        /* time at which key expires (or 0) */
```

```
@@ -85,14 +84,10 @@
```

```
gid_t            gid;
key_perm_t       perm;          /* access permissions */
unsigned short   quotalen;     /* length added to quota */
- unsigned short datalen;      /* payload data length */
- unsigned short flags;       /* status flags (change with lock writelocked) */
-#define KEY_FLAG_INSTANTIATED 0x00000001 /* set if key has been instantiated */
-#define KEY_FLAG_DEAD        0x00000002 /* set if key type has been deleted */
-#define KEY_FLAG_REVOKED     0x00000004 /* set if key had been revoked */
-#define KEY_FLAG_IN_QUOTA    0x00000008 /* set if key consumes quota */
-#define KEY_FLAG_USER_CONSTRUCT 0x00000010 /* set if key is being constructed in use */
-#define KEY_FLAG_NEGATIVE    0x00000020 /* set if key is negative */
+ unsigned short datalen;      /* payload data length
+ * - may not match RCU dereferenced payload
```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```

+                                     * - payload should contain own length
+                                     */

#ifdef KEY_DEBUGGING
    unsigned                magic;
@@ -100,6 +95,14 @@
#define KEY_DEBUG_MAGIC_X    0xf8e9dacbu
#endif

+    unsigned long          flags;                /* status flags (change with bitops) */
+#define KEY_FLAG_INSTANTIATED 0    /* set if key has been instantiated */
+#define KEY_FLAG_DEAD        1    /* set if key type has been deleted */
+#define KEY_FLAG_REVOKED    2    /* set if key had been revoked */
+#define KEY_FLAG_IN_QUOTA   3    /* set if key consumes quota */
+#define KEY_FLAG_USER_CONSTRUCT 4    /* set if key is being constructed in userspace */
+#define KEY_FLAG_NEGATIVE   5    /* set if key is negative */
+
+    /* the description string
+     * - this is used to match a key against search criteria
+     * - this should be a printable string
+
+@@ -249,6 +252,8 @@

extern struct key *key_lookup(key_serial_t id);

+extern void keyring_replace_payload(struct key *key, void *replacement);
+
#define key_serial(key) ((key) ? (key)->serial : 0)

/*
diff -uNr linux-2.6.11/include/linux/key-ui.h linux-2.6.11-keys-rcu/include/linux/key-ui.h
--- linux-2.6.11/include/linux/key-ui.h 2005-01-04 11:13:54.000000000 +0000
+++ linux-2.6.11-keys-rcu/include/linux/key-ui.h 2005-03-08 19:13:00.000000000 +0000
@@ -31,8 +31,10 @@
 * subscribed
 */
struct keyring_list {
-    unsigned    maxkeys;                /* max keys this list can hold */
-    unsigned    nkeys;                  /* number of keys currently held */
+    struct rcu_head rcu;                /* RCU deletion hook */
+    unsigned short maxkeys;            /* max keys this list can hold */
+    unsigned short nkeys;              /* number of keys currently held */
+    unsigned short delkey;             /* key to be unlinked by RCU */
    struct key    *keys[0];
};

diff -uNr linux-2.6.11/security/keys/key.c linux-2.6.11-keys-rcu/security/keys/key.c
--- linux-2.6.11/security/keys/key.c 2005-03-02 12:09:11.000000000 +0000
+++ linux-2.6.11-keys-rcu/security/keys/key.c 2005-03-09 16:49:04.375062714 +0000
@@ -293,7 +293,6 @@
    }

    atomic_set(&key->usage, 1);
-    rwlock_init(&key->lock);
    init_rwsem(&key->sem);
    key->type = type;
    key->user = user;
@@ -307,7 +306,7 @@
    key->payload.data = NULL;

    if (!not_in_quota)
-        key->flags |= KEY_FLAG_IN_QUOTA;
+        __set_bit(KEY_FLAG_IN_QUOTA, &key->flags);

```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```
memset(&key->type_data, 0, sizeof(key->type_data));

@@ -358,7 +357,7 @@
    key_check(key);

    /* contemplate the quota adjustment */
-   if (delta != 0 && key->flags & KEY_FLAG_IN_QUOTA) {
+   if (delta != 0 && test_bit(KEY_FLAG_IN_QUOTA, &key->flags)) {
        spin_lock(&key->user->lock);

        if (delta > 0 &&
@@ -404,23 +403,17 @@
        down_write(&key_construction_sem);

    /* can't instantiate twice */
-   if (!(key->flags & KEY_FLAG_INSTANTIATED)) {
+   if (!test_bit(KEY_FLAG_INSTANTIATED, &key->flags)) {
        /* instantiate the key */
        ret = key->type->instantiate(key, data, datalen);

        if (ret == 0) {
            /* mark the key as being instantiated */
-           write_lock(&key->lock);
-
-           atomic_inc(&key->user->nikeys);
-           key->flags |= KEY_FLAG_INSTANTIATED;
+           set_bit(KEY_FLAG_INSTANTIATED, &key->flags);

-           if (key->flags & KEY_FLAG_USER_CONSTRUCT) {
-               key->flags &= ~KEY_FLAG_USER_CONSTRUCT;
+           if (test_and_clear_bit(KEY_FLAG_USER_CONSTRUCT, &key->flags))
                awoken = 1;
-           }

-           write_unlock(&key->lock);

            /* and link it into the destination keyring */
            if (keyring)
@@ -485,21 +478,17 @@
        down_write(&key_construction_sem);

    /* can't instantiate twice */
-   if (!(key->flags & KEY_FLAG_INSTANTIATED)) {
+   if (!test_bit(KEY_FLAG_INSTANTIATED, &key->flags)) {
        /* mark the key as being negatively instantiated */
-       write_lock(&key->lock);
-
-       atomic_inc(&key->user->nikeys);
-       key->flags |= KEY_FLAG_INSTANTIATED | KEY_FLAG_NEGATIVE;
+       set_bit(KEY_FLAG_NEGATIVE, &key->flags);
+       set_bit(KEY_FLAG_INSTANTIATED, &key->flags);
        now = current_kernel_time();
        key->expiry = now.tv_sec + timeout;

-       if (key->flags & KEY_FLAG_USER_CONSTRUCT) {
-           key->flags &= ~KEY_FLAG_USER_CONSTRUCT;
+       if (test_and_clear_bit(KEY_FLAG_USER_CONSTRUCT, &key->flags))
                awoken = 1;
-       }

-       write_unlock(&key->lock);
```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```
        ret = 0;

        /* and link it into the destination keyring */
@@ -552,8 +541,10 @@
        rb_erase(&key->serial_node, &key_serial_tree);
        spin_unlock(&key_serial_lock);

+       key_check(key);
+
        /* deal with the user's key tracking and quota */
-       if (key->flags & KEY_FLAG_IN_QUOTA) {
+       if (test_bit(KEY_FLAG_IN_QUOTA, &key->flags)) {
                spin_lock(&key->user->lock);
                key->user->qnkeys--;
                key->user->qnbytes -= key->quotalen;
@@ -561,7 +552,7 @@
        }

        atomic_dec(&key->user->nkeys);
-       if (key->flags & KEY_FLAG_INSTANTIATED)
+       if (test_bit(KEY_FLAG_INSTANTIATED, &key->flags))
                atomic_dec(&key->user->nkeys);

        key_user_put(key->user);
@@ -630,9 +621,9 @@
        goto error;

found:
-       /* pretend doesn't exist if it's dead */
+       /* pretend it doesn't exist if it's dead */
        if (atomic_read(&key->usage) == 0 ||
-       (key->flags & KEY_FLAG_DEAD) ||
+       test_bit(KEY_FLAG_DEAD, &key->flags) ||
        key->type == &key_type_dead)
                goto not_found;

@@ -707,12 +698,9 @@

        ret = key->type->update(key, payload, plen);

-       if (ret == 0) {
+       if (ret == 0)
                /* updating a negative key instantiates it */
                write_lock(&key->lock);
                key->flags &= ~KEY_FLAG_NEGATIVE;
                write_unlock(&key->lock);
-       }
+       clear_bit(KEY_FLAG_NEGATIVE, &key->flags);

        up_write(&key->sem);

@@ -840,12 +828,9 @@
        down_write(&key->sem);
        ret = key->type->update(key, payload, plen);

-       if (ret == 0) {
+       if (ret == 0)
                /* updating a negative key instantiates it */
                write_lock(&key->lock);
                key->flags &= ~KEY_FLAG_NEGATIVE;
                write_unlock(&key->lock);
-       }
```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```

+         clear_bit(KEY_FLAG_NEGATIVE, &key->flags);
+
+         up_write(&key->sem);
+     }
@@ -891,10 +876,7 @@
+         goto error2;
+
+         atomic_inc(&key->user->nikeys);
-
-     write_lock(&key->lock);
-     key->flags |= KEY_FLAG_INSTANTIATED;
-     write_unlock(&key->lock);
+     set_bit(KEY_FLAG_INSTANTIATED, &key->flags);

error_k:
    up_read(&key_types_sem);
@@ -921,9 +903,7 @@
    /* make sure no one's trying to change or use the key when we mark
     * it */
    down_write(&key->sem);
-     write_lock(&key->lock);
-     key->flags |= KEY_FLAG_REVOKED;
-     write_unlock(&key->lock);
+     set_bit(KEY_FLAG_REVOKED, &key->flags);
    up_write(&key->sem);

} /* end key_revoke() */
@@ -974,24 +954,33 @@
    /* withdraw the key type */
    list_del_init(&ktype->link);

-     /* need to withdraw all keys of this type */
+     /* mark all the keys of this type dead */
    spin_lock(&key_serial_lock);

    for (_n = rb_first(&key_serial_tree); _n; _n = rb_next(_n)) {
        key = rb_entry(_n, struct key, serial_node);

-         if (key->type != ktype)
-             continue;
+         if (key->type == ktype)
+             key->type = &key_type_dead;
+     }
+
+     spin_unlock(&key_serial_lock);
+
+     /* make sure everyone revalidates their keys */
+     synchronize_kernel();
+
+     /* we should now be able to destroy the payloads of all the keys of
+      * this type with impunity */
+     spin_lock(&key_serial_lock);

-         write_lock(&key->lock);
-         key->type = &key_type_dead;
-         write_unlock(&key->lock);
-
-         /* there shouldn't be anyone looking at the description or
-          * payload now */
-         if (ktype->destroy)
-             ktype->destroy(key);
-         memset(&key->payload, 0xbd, sizeof(key->payload));

```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```
+ for (_n = rb_first(&key_serial_tree); _n; _n = rb_next(_n)) {
+     key = rb_entry(_n, struct key, serial_node);
+
+     if (key->type == ktype) {
+         if (ktype->destroy)
+             ktype->destroy(key);
+         memset(&key->payload, 0xbd, sizeof(key->payload));
+     }
+ }
+
+ spin_unlock(&key_serial_lock);
diff -uNr linux-2.6.11/security/keys/keyctl.c linux-2.6.11-keys-rcu/security/keys/keyctl.c
--- linux-2.6.11/security/keys/keyctl.c 2005-03-02 12:09:11.000000000 +0000
+++ linux-2.6.11-keys-rcu/security/keys/keyctl.c 2005-03-08 19:19:19.000000000 +0000
@@ -728,7 +728,6 @@
     /* make the changes with the locks held to prevent chown/chown races */
     ret = -EACCES;
     down_write(&key->sem);
-    write_lock(&key->lock);

     if (!capable(CAP_SYS_ADMIN)) {
         /* only the sysadmin can chown a key to some other UID */
@@ -755,7 +754,6 @@
     ret = 0;

     no_access:
-    write_unlock(&key->lock);
     up_write(&key->sem);
     key_put(key);
     error:
@@ -784,26 +782,19 @@
     goto error;
 }

-    /* make the changes with the locks held to prevent chown/chmod
-     * races */
+    /* make the changes with the locks held to prevent chown/chmod races */
     ret = -EACCES;
     down_write(&key->sem);
-    write_lock(&key->lock);

-    /* if we're not the sysadmin, we can only chmod a key that we
-     * own */
-    if (!capable(CAP_SYS_ADMIN) && key->uid != current->fsuid)
-        goto no_access;

-    /* changing the permissions mask */
     key->perm = perm;
     ret = 0;
+    /* if we're not the sysadmin, we can only change a key that we own */
+    if (capable(CAP_SYS_ADMIN) || key->uid == current->fsuid) {
+        key->perm = perm;
+        ret = 0;
+    }

- no_access:
-    write_unlock(&key->lock);
     up_write(&key->sem);
     key_put(key);
- error:
+error:
     return ret;

```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```
    } /* end keyctl_setperm_key() */
diff -uNr linux-2.6.11/security/keys/keyring.c linux-2.6.11-keys-rcu/security/keys/keyring.c
--- linux-2.6.11/security/keys/keyring.c      2005-03-02 12:09:11.000000000 +0000
+++ linux-2.6.11-keys-rcu/security/keys/keyring.c    2005-03-09 16:49:16.506055715 +0000
@@ -132,10 +132,17 @@
        (PAGE_SIZE - sizeof(*klist)) / sizeof(struct key);

-       ret = 0;
-       sklist = source->payload.subscriptions;

-       if (sklist && sklist->nkeys > 0) {
+       /* find out how many keys are currently linked */
+       rcu_read_lock();
+       sklist = rcu_dereference(source->payload.subscriptions);
+       max = 0;
+       if (sklist)
+           max = sklist->nkeys;
+       rcu_read_unlock();
+
+       /* allocate a new payload and stuff load with key links */
+       if (max > 0) {
+           BUG_ON(max > limit);

+           max = (max + 3) & ~3;
@@ -148,6 +155,10 @@
+           if (!klist)
+               goto error;

+           /* set links */
+           rcu_read_lock();
+           sklist = rcu_dereference(source->payload.subscriptions);
+
+           klist->maxkeys = max;
+           klist->nkeys = sklist->nkeys;
+           memcpy(klist->keys,
@@ -157,7 +168,9 @@
+           for (loop = klist->nkeys - 1; loop >= 0; loop--)
+               atomic_inc(&klist->keys[loop]->usage);

-           keyring->payload.subscriptions = klist;
+           rcu_read_unlock();
+
+           rcu_assign_pointer(keyring->payload.subscriptions, klist);
+           ret = 0;
+       }

@@ -192,7 +205,7 @@
+       write_unlock(&keyring_name_lock);
+   }

-   klist = keyring->payload.subscriptions;
+   klist = rcu_dereference(keyring->payload.subscriptions);
+   if (klist) {
+       for (loop = klist->nkeys - 1; loop >= 0; loop--)
+           key_put(klist->keys[loop]);
@@ -216,17 +229,20 @@
+       seq_puts(m, "[anon]");
+   }

-   klist = keyring->payload.subscriptions;
+   rcu_read_lock();
```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```
+ klist = rcu_dereference(keyring->payload.subscriptions);
  if (klist)
      seq_printf(m, ": %u/%u", klist->nkeys, klist->maxkeys);
  else
      seq_puts(m, ": empty");
+ rcu_read_unlock();

} /* end keyring_describe() */

/*****
/*
 * read a list of key IDs from the keyring's contents
+ * - the keyring's semaphore is read-locked
 */
static long keyring_read(const struct key *keyring,
                        char __user *buffer, size_t buflen)
@@ -237,7 +253,7 @@
    int loop, ret;

    ret = 0;
- klist = keyring->payload.subscriptions;
+ klist = rcu_dereference(keyring->payload.subscriptions);

    if (klist) {
        /* calculate how much data we could return */
@@ -320,7 +336,7 @@
        key_match_func_t match)
    {
        struct {
-         struct key *keyring;
+         struct keyring_list *keylist;
            int kix;
        } stack[KEYRING_SEARCH_MAX_DEPTH];

@@ -328,10 +344,12 @@
        struct timespec now;
        struct key *key;
        long err;
- int sp, psp, kix;
+ int sp, kix;

        key_check(keyring);

+ rcu_read_lock();
+
        /* top keyring must have search permission to begin the search */
        key = ERR_PTR(-EACCES);
        if (!key_permission(keyring, KEY_SEARCH))
@@ -347,11 +365,10 @@
        /* start processing a new keyring */
    descend:
- read_lock(&keyring->lock);
- if (keyring->flags & KEY_FLAG_REVOKED)
+ if (test_bit(KEY_FLAG_REVOKED, &keyring->flags))
        goto not_this_keyring;

- keylist = keyring->payload.subscriptions;
+ keylist = rcu_dereference(keyring->payload.subscriptions);
    if (!keylist)
        goto not_this_keyring;
```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```
@@ -364,7 +381,7 @@
        continue;

        /* skip revoked keys and expired keys */
-       if (key->flags & KEY_FLAG_REVOKED)
+       if (test_bit(KEY_FLAG_REVOKED, &key->flags))
            continue;

        if (key->expiry && now.tv_sec >= key->expiry)
@@ -379,7 +396,7 @@
        continue;

        /* we set a different error code if we find a negative key */
-       if (key->flags & KEY_FLAG_NEGATIVE) {
+       if (test_bit(KEY_FLAG_NEGATIVE, &key->flags)) {
            err = -ENOKEY;
            continue;
        }
@@ -390,48 +407,37 @@
        /* search through the keyrings nested in this one */
        kix = 0;
    ascend:
-       while (kix < keylist->nkeys) {
+       for (; kix < keylist->nkeys; kix++) {
            key = keylist->keys[kix];
            if (key->type != &key_type_keyring)
-               goto next;
+               continue;

            /* recursively search nested keyrings
             * - only search keyrings for which we have search permission
             */
            if (sp >= KEYRING_SEARCH_MAX_DEPTH)
-               goto next;
+               continue;

            if (!key_permission(key, KEY_SEARCH))
-               goto next;
-
-             /* evade loops in the keyring tree */
-             for (psp = 0; psp < sp; psp++)
-                 if (stack[psp].keyring == keyring)
-                     goto next;
+               continue;

            /* stack the current position */
-            stack[sp].keyring = keyring;
+            stack[sp].keylist = keylist;
            stack[sp].kix = kix;
            sp++;

            /* begin again with the new keyring */
            keyring = key;
            goto descend;
-
-         next:
-             kix++;
        }

        /* the keyring we're looking at was disqualified or didn't contain a
         * matching key */
    not_this_keyring:
```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```
- read_unlock(&keyring->lock);
-
+   if (sp > 0) {
+       /* resume the processing of a keyring higher up in the tree */
+       sp--;
+       keyring = stack[sp].keyring;
+       keylist = keyring->payload.subscriptions;
+   +   keylist = stack[sp].keylist;
+       kix = stack[sp].kix + 1;
+       goto ascend;
+   }
@@ -442,16 +448,9 @@
+   /* we found a viable match */
+   found:
+       atomic_inc(&key->usage);
+       read_unlock(&keyring->lock);
+
+       /* unwind the keyring stack */
+       while (sp > 0) {
+           sp--;
+           read_unlock(&stack[sp].keyring->lock);
+       }
+
+       key_check(key);
+   error:
+   +   rcu_read_unlock();
+       return key;
+
+   } /* end keyring_search_aux() */
@@ -489,7 +488,9 @@
+   struct key *key;
+   int loop;
+
+   klist = keyring->payload.subscriptions;
+   rcu_read_lock();
+
+   +   klist = rcu_dereference(keyring->payload.subscriptions);
+   if (klist) {
+       for (loop = 0; loop < klist->nkeys; loop++) {
+           key = klist->keys[loop];
@@ -497,7 +498,7 @@
+           if (key->type == ktype &&
+               key->type->match(key, description) &&
+               key_permission(key, perm) &&
-               !(key->flags & KEY_FLAG_REVOKED)
+               !test_bit(KEY_FLAG_REVOKED, &key->flags)
+           )
+               goto found;
+       }
@@ -509,6 +510,7 @@
+   found:
+       atomic_inc(&key->usage);
+   error:
+   +   rcu_read_unlock();
+       return key;
+
+   } /* end __keyring_search_one() */
@@ -540,7 +542,7 @@
+
+           &keyring_name_hash[bucket],
+           type_data.link
+       ) {
-           if (keyring->flags & KEY_FLAG_REVOKED)
```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```
+             if (test_bit(KEY_FLAG_REVOKED, &keyring->flags))
+                 continue;

+             if (strcmp(keyring->description, name) != 0)
@@ -579,7 +581,7 @@
static int keyring_detect_cycle(struct key *A, struct key *B)
{
    struct {
-        struct key *subtree;
+        struct keyring_list *keylist;
+        int kix;
    } stack[KEYRING_SEARCH_MAX_DEPTH];

@@ -587,20 +589,21 @@
    struct key *subtree, *key;
    int sp, kix, ret;

+    rcu_read_lock();
+
    ret = -EDEADLK;
    if (A == B)
-        goto error;
+        goto cycle_detected;

    subtree = B;
    sp = 0;

    /* start processing a new keyring */
descend:
-    read_lock(&subtree->lock);
-    if (subtree->flags & KEY_FLAG_REVOKED)
+    if (test_bit(KEY_FLAG_REVOKED, &subtree->flags))
+        goto not_this_keyring;

-    keylist = subtree->payload.subscriptions;
+    keylist = rcu_dereference(subtree->payload.subscriptions);
+    if (!keylist)
+        goto not_this_keyring;
    kix = 0;
@@ -619,7 +622,7 @@
+
+        goto too_deep;

+        /* stack the current position */
-        stack[sp].subtree = subtree;
+        stack[sp].keylist = keylist;
+        stack[sp].kix = kix;
+        sp++;

@@ -632,13 +635,10 @@
    /* the keyring we're looking at was disqualified or didn't contain a
    * matching key */
not_this_keyring:
-    read_unlock(&subtree->lock);
-
+    if (sp > 0) {
+        /* resume the checking of a keyring higher up in the tree */
+        sp--;
-        subtree = stack[sp].subtree;
-        keylist = subtree->payload.subscriptions;
+        keylist = stack[sp].keylist;
+        kix = stack[sp].kix + 1;
+        goto ascend;

```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```

    }
@@ -646,30 +646,36 @@
    ret = 0; /* no cycles detected */

    error:
+   rcu_read_unlock();
    return ret;

    too_deep:
        ret = -ELOOP;
-   goto error_unwind;
+   goto error;
+
    cycle_detected:
        ret = -EDEADLK;
- error_unwind:
-   read_unlock(&subtree->lock);
-
-   /* unwind the keyring stack */
-   while (sp > 0) {
-       sp--;
-       read_unlock(&stack[sp].subtree->lock);
-   }
-
    goto error;

} /* end keyring_detect_cycle() */

/*****
/*
+ * dispose of a keyring list after the RCU grace period
+ */
+static void keyring_link_rcu_disposal(struct rcu_head *rcu)
+{
+   struct keyring_list *klist =
+       container_of(rcu, struct keyring_list, rcu);
+
+   kfree(klist);
+} /* end keyring_link_rcu_disposal() */
+
+*****/
+/*
+ * link a key into to a keyring
+ * - must be called with the keyring's semaphore held
+ * - must be called with the keyring's semaphore write-locked
+ */
int __key_link(struct key *keyring, struct key *key)
{
@@ -679,7 +685,7 @@
    int ret;

    ret = -EKEYREVOKED;
-   if (keyring->flags & KEY_FLAG_REVOKED)
+   if (test_bit(KEY_FLAG_REVOKED, &keyring->flags))
        goto error;

    ret = -ENOTDIR;
@@ -704,25 +710,31 @@
    if (ret < 0)
        goto error2;

```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```
+ preempt_disable();
+ klist = keyring->payload.subscriptions;

+ if (klist && klist->nkeys < klist->maxkeys) {
+     /* there's sufficient slack space to add directly */
+     atomic_inc(&key->usage);

-     write_lock(&keyring->lock);
-     klist->keys[klist->nkeys++] = key;
-     write_unlock(&keyring->lock);
+     klist->keys[klist->nkeys] = key;
+     wmb();
+     klist->nkeys++;
+     preempt_enable();

+     ret = 0;
+ }
+ else {
+     preempt_enable();
+
+     /* grow the key list */
+     max = 4;
+     if (klist)
+         max += klist->maxkeys;

+     ret = -ENFILE;
+     if (max > 65535)
+         goto error3;
+     size = sizeof(*klist) + sizeof(*key) * max;
+     if (size > PAGE_SIZE)
+         goto error3;
@@ -743,14 +755,13 @@

+     /* add the key into the new space */
+     atomic_inc(&key->usage);

-     write_lock(&keyring->lock);
-     keyring->payload.subscriptions = nklist;
-     nklist->keys[nklist->nkeys++] = key;
-     write_unlock(&keyring->lock);
+     rcu_assign_pointer(keyring->payload.subscriptions, nklist);

+     /* dispose of the old keyring list */
+     kfree(klist);
+     if (klist)
+         call_rcu(&klist->rcu, keyring_link_rcu_disposal);

+     ret = 0;
+ }
@@ -791,11 +802,26 @@

+ /*****
+ /*
+ * dispose of a keyring list after the RCU grace period, freeing the unlinked
+ * key
+ */
+static void keyring_unlink_rcu_disposal(struct rcu_head *rcu)
+{
+     struct keyring_list *klist =
+         container_of(rcu, struct keyring_list, rcu);
+ }
```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```
+     key_put(klist->keys[klist->delkey]);
+     kfree(klist);
+
+} /* end keyring_unlink_rcu_disposal() */
+
+/*
+*****
+*/
+ * unlink the first link to a key from a keyring
+ */
int key_unlink(struct key *keyring, struct key *key)
{
-     struct keyring_list *klist;
+     struct keyring_list *klist, *nklist;
+     int loop, ret;

+     key_check(keyring);
@@ -819,31 +845,45 @@
+     ret = -ENOENT;
+     goto error;

- key_is_present:
+key_is_present:
+     /* we need to copy the key list for RCU purposes */
+     nklist = kmalloc(sizeof(*klist) + sizeof(*key) * klist->maxkeys,
+                     GFP_KERNEL);
+     if (!nklist)
+         goto nomem;
+     nklist->maxkeys = klist->maxkeys;
+     nklist->nkeys = klist->nkeys - 1;
+
+     if (loop > 0)
+         memcpy(&nklist->keys[0],
+                &klist->keys[0],
+                loop * sizeof(klist->keys[0]));
+
+     if (loop < nklist->nkeys)
+         memcpy(&nklist->keys[loop],
+                &klist->keys[loop + 1],
+                (nklist->nkeys - loop) * sizeof(klist->keys[0]));
+
+     /* adjust the user's quota */
+     key_payload_reserve(keyring,
+                         keyring->datalen - KEYQUOTA_LINK_BYTES);

-     /* shuffle down the key pointers
-      * - it might be worth shrinking the allocated memory, but that runs
-      *   the risk of ENOMEM as we would have to copy
-      */
-     write_lock(&keyring->lock);
-
-     klist->nkeys--;
-     if (loop < klist->nkeys)
-         memcpy(&klist->keys[loop],
-                &klist->keys[loop + 1],
-                (klist->nkeys - loop) * sizeof(struct key *));
-
-     write_unlock(&keyring->lock);
+     rcu_assign_pointer(keyring->payload.subscriptions, nklist);

+     up_write(&keyring->sem);
-     key_put(key);
+
+}
```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```

+     /* schedule for later cleanup */
+     klist->delkey = loop;
+     call_rcu(&klist->rcu, keyring_unlink_rcu_disposal);
+
+     ret = 0;

- error:
+error:
+     return ret;

+nomem:
+     ret = -ENOMEM;
+     up_write(&keyring->sem);
+     goto error;

} /* end key_unlink() */

@@ -851,13 +891,32 @@

/*****
/*
+ * dispose of a keyring list after the RCU grace period, releasing the keys it
+ * links to
+ */
+static void keyring_clear_rcu_disposal(struct rcu_head *rcu)
+{
+     struct keyring_list *klist;
+     int loop;
+
+     klist = container_of(rcu, struct keyring_list, rcu);
+
+     for (loop = klist->nkeys - 1; loop >= 0; loop--)
+         key_put(klist->keys[loop]);
+
+     kfree(klist);
+
+} /* end keyring_clear_rcu_disposal() */
+
+*****/
+/*
+ * clear the specified process keyring
+ * - implements keyctl(KEYCTL_CLEAR)
+ */
int keyring_clear(struct key *keyring)
{
+     struct keyring_list *klist;
-     int loop, ret;
+     int ret;

+     ret = -ENOTDIR;
+     if (keyring->type == &key_type_keyring) {
@@ -870,20 +929,15 @@
+         key_payload_reserve(keyring,
+                             sizeof(struct keyring_list));

-         write_lock(&keyring->lock);
-         keyring->payload.subscriptions = NULL;
-         write_unlock(&keyring->lock);
+         rcu_assign_pointer(keyring->payload.subscriptions,
+                             NULL);
+     }

+     up_write(&keyring->sem);

```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```

        /* free the keys after the locks have been dropped */
-       if (klist) {
-           for (loop = klist->nkeys - 1; loop >= 0; loop--)
-               key_put(klist->keys[loop]);
-
-           kfree(klist);
-       }
+       if (klist)
+           call_rcu(&klist->rcu, keyring_clear_rcu_disposal);

        ret = 0;
    }
diff -uNr linux-2.6.11/security/keys/proc.c linux-2.6.11-keys-rcu/security/keys/proc.c
--- linux-2.6.11/security/keys/proc.c      2005-01-04 11:14:01.000000000 +0000
+++ linux-2.6.11-keys-rcu/security/keys/proc.c  2005-03-09 16:49:46.284583752 +0000
@@ -140,7 +140,7 @@

        now = current_kernel_time();

-       read_lock(&key->lock);
+       rcu_read_lock();

        /* come up with a suitable timeout value */
        if (key->expiry == 0) {
@@ -164,14 +164,17 @@
                sprintf(xbuf, "%luw", timo / (60*60*24*7));
        }

+#define showflag(KEY, LETTER, FLAG) \
+    (test_bit(FLAG, &(KEY)->flags) ? LETTER : '-')
+
        seq_printf(m, "%08x %c%c%c%c%c%c %5d %4s %06x %5d %5d %-.9s ",
                key->serial,
-               key->flags & KEY_FLAG_INSTANTIATED ? 'I' : '-',
-               key->flags & KEY_FLAG_REVOKED      ? 'R' : '-',
-               key->flags & KEY_FLAG_DEAD         ? 'D' : '-',
-               key->flags & KEY_FLAG_IN_QUOTA     ? 'Q' : '-',
-               key->flags & KEY_FLAG_USER_CONSTRUCT ? 'U' : '-',
-               key->flags & KEY_FLAG_NEGATIVE     ? 'N' : '-',
+               showflag(key, 'I', KEY_FLAG_INSTANTIATED),
+               showflag(key, 'R', KEY_FLAG_REVOKED),
+               showflag(key, 'D', KEY_FLAG_DEAD),
+               showflag(key, 'Q', KEY_FLAG_IN_QUOTA),
+               showflag(key, 'U', KEY_FLAG_USER_CONSTRUCT),
+               showflag(key, 'N', KEY_FLAG_NEGATIVE),
                atomic_read(&key->usage),
                xbuf,
                key->perm,
@@ -179,11 +179,13 @@
                key->gid,
                key->type->name);

+#undef showflag
+
        if (key->type->describe)
            key->type->describe(key, m);
        seq_putc(m, '\n');

-       read_unlock(&key->lock);
+       rcu_read_unlock();

```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```
return 0;

diff -uNr linux-2.6.11/security/keys/process_keys.c linux-2.6.11-keys-rcu/security/keys/process_k
--- linux-2.6.11/security/keys/process_keys.c 2005-01-04 11:14:01.000000000 +0000
+++ linux-2.6.11-keys-rcu/security/keys/process_keys.c 2005-03-09 16:49:40.405071823 +0000
@@ -38,10 +38,9 @@
     .serial          = 2,
     .type            = &key_type_keyring,
     .user            = &root_key_user,
-    .lock            = RW_LOCK_UNLOCKED,
     .sem             = __RWSEM_INITIALIZER(root_user_keyring.sem),
     .perm            = KEY_USR_ALL,
-    .flags            = KEY_FLAG_INSTANTIATED,
+    .flags            = 1 << KEY_FLAG_INSTANTIATED,
     .description     = "_uid.0",
 #ifdef KEY_DEBUGGING
     .magic            = KEY_DEBUG_MAGIC,
@@ -54,10 +53,9 @@
     .serial          = 1,
     .type            = &key_type_keyring,
     .user            = &root_key_user,
-    .lock            = RW_LOCK_UNLOCKED,
     .sem             = __RWSEM_INITIALIZER(root_session_keyring.sem),
     .perm            = KEY_USR_ALL,
-    .flags            = KEY_FLAG_INSTANTIATED,
+    .flags            = 1 << KEY_FLAG_INSTANTIATED,
     .description     = "_uid_ses.0",
 #ifdef KEY_DEBUGGING
     .magic            = KEY_DEBUG_MAGIC,
@@ -317,18 +315,14 @@
     /* update the ownership of the process keyring */
     if (tsk->process_keyring) {
         down_write(&tsk->process_keyring->sem);
-        write_lock(&tsk->process_keyring->lock);
         tsk->process_keyring->uid = tsk->fsuid;
-        write_unlock(&tsk->process_keyring->lock);
         up_write(&tsk->process_keyring->sem);
     }

     /* update the ownership of the thread keyring */
     if (tsk->thread_keyring) {
         down_write(&tsk->thread_keyring->sem);
-        write_lock(&tsk->thread_keyring->lock);
         tsk->thread_keyring->uid = tsk->fsuid;
-        write_unlock(&tsk->thread_keyring->lock);
         up_write(&tsk->thread_keyring->sem);
     }

@@ -343,18 +337,14 @@
     /* update the ownership of the process keyring */
     if (tsk->process_keyring) {
         down_write(&tsk->process_keyring->sem);
-        write_lock(&tsk->process_keyring->lock);
         tsk->process_keyring->gid = tsk->fsgid;
-        write_unlock(&tsk->process_keyring->lock);
         up_write(&tsk->process_keyring->sem);
     }

     /* update the ownership of the thread keyring */
     if (tsk->thread_keyring) {
         down_write(&tsk->thread_keyring->sem);
-        write_lock(&tsk->thread_keyring->lock);
```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```
        tsk->thread_keyring->gid = tsk->fsgid;
-       write_unlock(&tsk->thread_keyring->lock);
        up_write(&tsk->thread_keyring->sem);
    }

@@ -565,7 +555,7 @@
}

    ret = -EIO;
-    if (!partial && !(key->flags & KEY_FLAG_INSTANTIATED))
+    if (!partial && !test_bit(KEY_FLAG_INSTANTIATED, &key->flags))
        goto invalid_key;

    ret = -EACCES;
diff -uNr linux-2.6.11/security/keys/request_key.c linux-2.6.11-keys-rcu/security/keys/request_key.c
--- linux-2.6.11/security/keys/request_key.c      2005-01-04 11:14:01.000000000 +0000
+++ linux-2.6.11-keys-rcu/security/keys/request_key.c  2005-03-09 16:49:29.924941798 +0000
@@ -94,7 +94,7 @@
    struct key_construction cons;
    struct timespec now;
    struct key *key;
-    int ret, negative;
+    int ret, negated;

    /* create a key and add it to the queue */
    key = key_alloc(type, description,
@@ -102,9 +102,7 @@
    if (IS_ERR(key))
        goto alloc_failed;

-    write_lock(&key->lock);
-    key->flags |= KEY_FLAG_USER_CONSTRUCT;
-    write_unlock(&key->lock);
+    set_bit(KEY_FLAG_USER_CONSTRUCT, &key->flags);

    cons.key = key;
    list_add_tail(&cons.link, &key->user->consq);
@@ -119,7 +117,7 @@

    /* if the key wasn't instantiated, then we want to give an error */
    ret = -ENOKEY;
-    if (!(key->flags & KEY_FLAG_INSTANTIATED))
+    if (!test_bit(KEY_FLAG_INSTANTIATED, &key->flags))
        goto request_failed;

    down_write(&key_construction_sem);
@@ -128,7 +126,7 @@

    /* also give an error if the key was negatively instantiated */
check_not_negative:
-    if (key->flags & KEY_FLAG_NEGATIVE) {
+    if (test_bit(KEY_FLAG_NEGATIVE, &key->flags)) {
        key_put(key);
        key = ERR_PTR(-ENOKEY);
    }
@@ -141,24 +139,23 @@
    * - remove from construction queue
    * - mark the key as dead
    */
-    negative = 0;
+    negated = 0;
    down_write(&key_construction_sem);
```



## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```
static int user_instantiate(struct key *key, const void *data, size_t datalen)
{
+   struct user_key_payload *upayload;
+   int ret;

    ret = -EINVAL;
@@ -58,13 +65,15 @@
    if (ret < 0)
        goto error;

-   /* attach the data */
    ret = -ENOMEM;
-   key->payload.data = kmalloc(datalen, GFP_KERNEL);
-   if (!key->payload.data)
+   upayload = kmalloc(sizeof(*upayload) + datalen, GFP_KERNEL);
+   if (!upayload)
        goto error;

-   memcpy(key->payload.data, data, datalen);
+   /* attach the data */
+   upayload->datalen = datalen;
+   memcpy(upayload->data, data, datalen);
+   rcu_assign_pointer(key->payload.data, upayload);
    ret = 0;

error:
@@ -75,18 +84,25 @@
/*****
/*
 * duplicate a user defined key
+ * - both keys' semaphores are locked against further modification
+ * - the new key cannot yet be accessed
 */
static int user_duplicate(struct key *key, const struct key *source)
{
+   struct user_key_payload *upayload, *spayload;
+   int ret;

    /* just copy the payload */
    ret = -ENOMEM;
-   key->payload.data = kmalloc(source->datalen, GFP_KERNEL);
+   upayload = kmalloc(sizeof(*upayload) + source->datalen, GFP_KERNEL);
+   if (upayload) {
+       spayload = rcu_dereference(source->payload.data);
+       BUG_ON(source->datalen != spayload->datalen);

-   if (key->payload.data) {
-       key->datalen = source->datalen;
-       memcpy(key->payload.data, source->payload.data, source->datalen);
+       upayload->datalen = key->datalen = spayload->datalen;
+       memcpy(upayload->data, spayload->data, key->datalen);
+
+       key->payload.data = upayload;
        ret = 0;
    }

@@ -96,40 +112,54 @@

/*****
/*
+ * dispose of the old data from an updated user defined key
+ */
```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```
+static void user_update_rcu_disposal(struct rcu_head *rcu)
+{
+    struct user_key_payload *upayload;
+
+    upayload = container_of(rcu, struct user_key_payload, rcu);
+
+    kfree(upayload);
+
+} /* end user_update_rcu_disposal() */
+
+/*
+ * update a user defined key
+ * - the key's semaphore is write-locked
+ */
static int user_update(struct key *key, const void *data, size_t datalen)
{
-    void *new, *zap;
+    struct user_key_payload *upayload, *zap;
    int ret;

    ret = -EINVAL;
    if (datalen <= 0 || datalen > 32767 || !data)
        goto error;

-    /* copy the data */
+    /* construct a replacement payload */
    ret = -ENOMEM;
-    new = kmalloc(datalen, GFP_KERNEL);
-    if (!new)
+    upayload = kmalloc(sizeof(*upayload) + datalen, GFP_KERNEL);
+    if (!upayload)
        goto error;

-    memcpy(new, data, datalen);
+    memcpy(upayload, data, datalen);
+    memcpy(upayload, data, datalen);

    /* check the quota and attach the new data */
-    zap = new;
-    write_lock(&key->lock);
+    zap = upayload;

    ret = key_payload_reserve(key, datalen);

    if (ret == 0) {
        /* attach the new data, displacing the old */
-        zap = key->payload.data;
-        key->payload.data = new;
+        zap = rcu_dereference(key->payload.data);
+        rcu_assign_pointer(key->payload.data, upayload);
        key->expiry = 0;
    }

-    write_unlock(&key->lock);
-    kfree(zap);
+    call_rcu(&zap->rcu, user_update_rcu_disposal);

error:
    return ret;
@@ -152,13 +182,15 @@
 */
```

## Linux-Kernel: [PATCH] keys: Discard key spinlock and use RCU for key payload

```
static void user_destroy(struct key *key)
{
-   kfree(key->payload.data);
+   struct user_key_payload *upayload = key->payload.data;
+
+   kfree(upayload);
} /* end user_destroy() */

/*****
/*
- * describe the user
+ * describe the user key
*/
static void user_describe(const struct key *key, struct seq_file *m)
{
@@ -171,18 +203,23 @@
/*****
/*
* read the key data
+ * - the key's semaphore is read-locked
*/
static long user_read(const struct key *key,
                     char __user *buffer, size_t buflen)
{
-   long ret = key->datalen;
+   struct user_key_payload *upayload;
+   long ret;
+
+   upayload = rcu_dereference(key->payload.data);
+   ret = upayload->datalen;

    /* we can return the data as is */
    if (buffer && buflen > 0) {
-       if (buflen > key->datalen)
-           buflen = key->datalen;
+       if (buflen > upayload->datalen)
+           buflen = upayload->datalen;

-       if (copy_to_user(buffer, key->payload.data, buflen) != 0)
+       if (copy_to_user(buffer, upayload->data, buflen) != 0)
            ret = -EFAULT;
    }
}
-

```

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@vger.kernel.org  
More majordomo info at <http://vger.kernel.org/majordomo-info.html>  
Please read the FAQ at <http://www.tux.org/lkml/>