

[PATCH] Futex waiters take an mm or inode reference

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2003-09/7592.html>

From: Jamie Lokier (jamie_at_shareable.org)

Date: 09/30/03

Date: Tue, 30 Sep 2003 08:42:46 +0100
To: torvalds@osdl.org

Patch: `futex_refs-2.6.0-test6`

This patch fixes an important hole in 2.6.0-test6 futexes. At the moment, a program can use FUTEX_FD to create futexes on mm's or inodes which are recycled, and thereby steal wakeups from unrelated tasks.

Rusty Russell wrote:

> *But why not solve the problem by just holding an mm reference, too?*

Exactly. With this patch each futex hold a reference to the mm or inode that it is keyed on.

Concerning FUTEX_REQUEUE:

> *Why not make the code a *whole* lot more readable (and only marginally slower, if at all) by doing it in two passes: pull them off onto a (on-stack) list in one pass, then requeue them all in another.*

Rusty will be glad that I have reimplemented `futex_requeue()` exactly as suggested: in two passes. This also removes that unpleasant double-lock for the two hash buckets: now only one at a time is locked.

The `set_current_state()` logic in `FUTEX_WAIT` has been simplified too.

Linus, please apply.

Thanks all,
— Jamie

```
diff -urN --exclude-from=dontdiff orig-2.6.0-test6/kernel/futex.c dual-2.6.0-test6/kernel/futex.c
--- orig-2.6.0-test6/kernel/futex.c 2003-09-30 05:41:14.000000000 +0100
+++ dual-2.6.0-test6/kernel/futex.c 2003-09-30 06:43:17.000000000 +0100
@@ -45,6 +45,9 @@
```

- * Futexes are matched on equal values of this key.
- * The key type depends on whether it's a shared or private mapping.
- * Don't rearrange members without looking at `hash_futex()`.

Linux-Kernel: [PATCH] Futex waiters take an mm or inode reference

```
+ *
+ * offset is aligned to a multiple of sizeof(u32) (== 4) by definition.
+ * We set bit 0 to indicate if it's an inode-based key.
+ */
union futex_key {
    struct {
@@ -172,9 +175,10 @@
    }

    /*
- * Linear mappings are also simple.
+ * Linear file mappings are also simple.
+ */
    key->shared.inode = vma->vm_file->f_dentry->d_inode;
+ key->both.offset++; /* Bit 0 of offset indicates inode-based key. */
    if (likely(!(vma->vm_flags & VM_NONLINEAR))) {
        key->shared.pgoff = (((uaddr - vma->vm_start) >> PAGE_SHIFT)
            + vma->vm_pgoff);
@@ -214,16 +218,55 @@
    return err;
}

+/*
+ * Take a reference to the resource addressed by a key.
+ *
+ * NOTE: mmap_sem MUST be held between get_futex_key() and calling this
+ * function, if it is called at all. mmap_sem keeps key->shared.inode valid.
+ */
+static inline void get_key_refs(union futex_key *key)
+{
+ if (key->both.ptr != 0) {
+ if (key->both.offset & 1)
+ atomic_inc(&key->shared.inode->i_count);
+ else
+ atomic_inc(&key->private.mm->mm_count);
+ }
+ }
+
+/*
+ * Drop a reference to the resource addressed by a key.
+ * The hash bucket lock must not be held.
+ */
+static inline void drop_key_refs(union futex_key *key)
+{
+ if (key->both.ptr != 0) {
+ if (key->both.offset & 1)
+ iput(key->shared.inode);
+ else
+ mmdrop(key->private.mm);
+ }
+ }
```

Linux-Kernel: [PATCH] Futex waiters take an mm or inode reference

```
+
+/* The hash bucket lock must be held when this is called. */
+static inline void wake_futex(struct futex_q *q)
+{
+ list_del_init(&q->list);
+ wake_up_all(&q->waiters);
+ if (q->filp)
+ send_sigio(&q->filp->f_owner, q->fd, POLL_IN);
+}

/*
 * Wake up all waiters hashed on the physical page that is mapped
 * to this virtual address:
 */
-static int futex_wake(unsigned long uaddr, int num)
+static int futex_wake(unsigned long uaddr, int nr_wake)
{
- struct list_head *i, *next, *head;
- struct futex_hash_bucket *bh;
  union futex_key key;
+ struct futex_hash_bucket *bh;
+ struct list_head *head;
+ struct futex_q *this, *next;
  int ret;

  down_read(&current->mm->mmap_sem);
@@ -236,21 +279,15 @@
  spin_lock(&bh->lock);
  head = &bh->chain;

- list_for_each_safe(i, next, head) {
- struct futex_q *this = list_entry(i, struct futex_q, list);
-
+ list_for_each_entry_safe(this, next, head, list) {
+   if (match_futex (&this->key, &key)) {
- list_del_init(i);
- wake_up_all(&this->waiters);
- if (this->filp)
- send_sigio(&this->filp->f_owner, this->fd, POLL_IN);
- ret++;
- if (ret >= num)
+ wake_futex(this);
+ if (++ret >= nr_wake)
+   break;
+   }
+ }
- spin_unlock(&bh->lock);

+ spin_unlock(&bh->lock);
out:
  up_read(&current->mm->mmap_sem);
```

Linux-Kernel: [PATCH] Futex waiters take an mm or inode reference

```
    return ret;
@@ -263,9 +300,10 @@
static int futex_requeue(unsigned long uaddr1, unsigned long uaddr2,
                        int nr_wake, int nr_requeue)
{
- struct list_head *i, *next, *head1, *head2;
- struct futex_hash_bucket *bh1, *bh2;
  union futex_key key1, key2;
+ struct futex_hash_bucket *bh;
+ struct list_head *head, moved;
+ struct futex_q *this, *next;
  int ret;

  down_read(&current->mm->mmap_sem);
@@ -277,65 +315,65 @@
  if (unlikely(ret != 0))
    goto out;

- bh1 = hash_futex(&key1);
- bh2 = hash_futex(&key2);
- if (bh1 < bh2) {
- spin_lock(&bh1->lock);
- spin_lock(&bh2->lock);
- } else {
- spin_lock(&bh2->lock);
- if (bh1 > bh2)
- spin_lock(&bh1->lock);
- }
- head1 = &bh1->chain;
- head2 = &bh2->chain;
-
- list_for_each_safe(i, next, head1) {
- struct futex_q *this = list_entry(i, struct futex_q, list);
+ bh = hash_futex(&key1);
+ spin_lock(&bh->lock);
+ head = &bh->chain;

+ INIT_LIST_HEAD(&moved);
+ list_for_each_entry_safe(this, next, head, list) {
  if (match_futex (&this->key, &key1)) {
- list_del_init(i);
    if (++ret <= nr_wake) {
- wake_up_all(&this->waiters);
- if (this->filp)
- send_sigio(&this->filp->f_owner,
- this->fd, POLL_IN);
+ wake_futex(this);
    } else {
- list_add_tail(i, head2);
- this->key = key2;
+ list_move_tail(&this->list, &moved);
```

Linux-Kernel: [PATCH] Futex waiters take an mm or inode reference

```

        if (ret - nr_wake >= nr_requeue)
            break;
    - /* Make sure to stop if key1 == key2 */
    - if (head1 == head2 && head1 != next)
    - head1 = i;
        }
    }
}
- if (bh1 < bh2) {
- spin_unlock(&bh2->lock);
- spin_unlock(&bh1->lock);
- } else {
- if (bh1 > bh2)
- spin_unlock(&bh1->lock);
- spin_unlock(&bh2->lock);
+
+ spin_unlock(&bh->lock);
+
+ if (!list_empty(&moved)) {
+ bh = hash_futex(&key2);
+ head = &bh->chain;
+ list_for_each_entry_safe(this, next, &moved, list) {
+ /* Don't hold the spinlock during drop_key_refs(). */
+ drop_key_refs(&this->key);
+ this->key = key2;
+ get_key_refs(&this->key);
+
+ spin_lock(&bh->lock);
+ list_add_tail(&this->list, head);
+ spin_unlock(&bh->lock);
+ }
    }
+
out:
    up_read(&current->mm->mmap_sem);
    return ret;
}

-static inline void queue_me(struct futex_q *q, union futex_key *key,
- int fd, struct file *filp)
+/*
+ * queue_me and unqueue_me must be called as a pair, each
+ * exactly once. They are called with the hashed spinlock held.
+ */
+
+ /* The key must be already stored in q->key. */
+static inline void queue_me(struct futex_q *q, int fd, struct file *filp)
{
- struct futex_hash_bucket *bh = hash_futex(key);
- struct list_head *head = &bh->chain;
+ struct futex_hash_bucket *bh;

```

Linux-Kernel: [PATCH] Futex waiters take an mm or inode reference

```
- q->key = *key;
  q->fd = fd;
  q->filp = filp;

+ init_waitqueue_head(&q->waiters);
+
+ get_key_refs(&q->key);
+ bh = hash_futex(&q->key);
+
  spin_lock(&bh->lock);
- list_add_tail(&q->list, head);
+ list_add_tail(&q->list, &bh->chain);
  spin_unlock(&bh->lock);
}

@@ -351,6 +389,8 @@
    ret = 1;
  }
  spin_unlock(&bh->lock);
+
+ drop_key_refs(&q->key);
  return ret;
}

@@ -358,19 +398,16 @@
{
  DECLARE_WAITQUEUE(wait, current);
  int ret, curval;
- union futex_key key;
  struct futex_q q;
- struct futex_hash_bucket *bh = NULL;
-
- init_waitqueue_head(&q.waiters);
+ struct futex_hash_bucket *bh;

  down_read(&current->mm->mmap_sem);

- ret = get_futex_key(uaddr, &key);
+ ret = get_futex_key(uaddr, &q.key);
  if (unlikely(ret != 0))
    goto out_release_sem;

- queue_me(&q, &key, -1, NULL);
+ queue_me(&q, -1, NULL);

  /*
   * Access the page after the futex is queued.
@@ -401,22 +438,18 @@
  * wakes us up.
  */
```

Linux-Kernel: [PATCH] Futex waiters take an mm or inode reference

```
    add_wait_queue(&q.waiters, &wait);
- bh = hash_futex(&key);
+ bh = hash_futex(&q.key);
    spin_lock(&bh->lock);
- set_current_state(TASK_INTERRUPTIBLE);

    if (unlikely(list_empty(&q.list))) {
- /*
- * We were woken already.
- */
+ /* We were woken already. */
        spin_unlock(&bh->lock);
- set_current_state(TASK_RUNNING);
        return 0;
    }

+ __set_current_state(TASK_INTERRUPTIBLE);
    spin_unlock(&bh->lock);
    time = schedule_timeout(time);
- set_current_state(TASK_RUNNING);

    /*
     * NOTE: we don't remove ourselves from the waitqueue because
@@ -446,7 +479,7 @@
    struct futex_q *q = filp->private_data;

    unqueue_me(q);
- kfree(filp->private_data);
+ kfree(q);
    return 0;
}

@@ -477,7 +510,6 @@
static int futex_fd(unsigned long uaddr, int signal)
{
    struct futex_q *q;
- union futex_key key;
    struct file *filp;
    int ret, err;

@@ -519,20 +551,22 @@
}

    down_read(&current->mm->mmap_sem);
- err = get_futex_key(uaddr, &key);
- up_read(&current->mm->mmap_sem);
+ err = get_futex_key(uaddr, &q->key);

    if (unlikely(err != 0)) {
+ up_read(&current->mm->mmap_sem);
        put_unused_fd(ret);
    }
}
```

Linux-Kernel: [PATCH] Futex waiters take an mm or inode reference

```
    put_filp(filp);
    kfree(q);
    return err;
}
```

```
- init_waitqueue_head(&q->waiters);
+ /* queue_me() must be called before releasing mmap_sem, because
+ key->shared.inode needs to be referenced while holding it. */
    filp->private_data = q;
```

```
- queue_me(q, &key, ret, filp);
+ queue_me(q, ret, filp);
+ up_read(&current->mm->mmap_sem);
```

```
    /* Now we map fd to filp, so userspace can access it */
    fd_install(ret, filp);
```

-

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/>