

[PATCH] 2.6.16 – futex: small optimization (?)

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-03/msg09494.html>

- *From:* Pierre PEIFFER <pierre.peiffer@xxxxxxxx>
 - *Date:* Tue, 28 Mar 2006 09:37:27 +0200
-

Hi,

I found a (optimization ?) problem in the futexes, during a futex_wake, if the waiter has a higher priority than the waker.

In fact, in this case, the waiter is immediately scheduled and tries to take a lock still held by the waker. This is specially expensive on UP or if both threads are on the same CPU, due to the two task-switchings. This produces an extra latency during a wakeup in pthread_cond_broadcast or pthread_cond_signal, for example.

See below my detailed explanation.

I found a solution given by the patch, at the end of this mail. It works for me on kernel 2.6.16, but the kernel hangs if I use it with -rt patch from Ingo Molnar. So, I have a doubt on the correctness of the patch.

The idea is simple: in unqueue_me, I first check
"if (list_empty(&q->list))"

If yes => we were woken (the list is initialized in wake_futex).
Then, it immediately returns and let the waker drop the key_refs (instead of the waiter).

=====
Here is the detailed explanation:

Imagine two threads, on UP or on the same CPU: a futex-waker (thread A)
and a futex-waiter (thread B); thread B having a higher priority, blocked and sleeping in futex_wait. Here is the scenario:

Thread A Thread B
(waker) (waiter with higher priority)

```
/* sleeping in futex_wait */
/* wake the futex */
/* (from futex_wake or futex_requeue) */
\_ wake_futex
\_ list_del_init(&q->list)
```

[PATCH] 2.6.16 – futex: small optimization (?)

```
\_ wake_up_all (thread B)
```

=> Thread B, due to its higher priority is immediately woken and scheduled
=> task-switch to thread B

```
/* sleeps */ /* awakes */
```

```
in futex_wait:
```

```
\_ ...  
\_ unqueue_me  
\_ lock_ptr = q->lock_ptr;  
\_ if (lock_ptr != 0) {  
/* TRUE */  
\_ spin_lock(lock_ptr);  
/* lock is still locked by the  
waker, thread A, in either  
futex_wake or futex_requeue  
*/
```

=> task-switch to the lock owner, thread A

```
/* awakes */ /* sleeps */
```

```
\_ q->lock_ptr = NULL;  
/* back to futex_wake or futex_requeue */  
\_ ...  
\_ spin_unlock(&bh->lock);  
/* this is q->lock_ptr in thread B */  
/* => waiters are woken */
```

=> task-switch to the lock waiter, thread B

```
/* sleeps */ /* awakes */
```

```
\_ if (lock_ptr != q->lock_ptr) {  
/* unfortunately, this is true */  
/* q->lock_ptr is now NULL */  
\_ spin_unlock(lock_ptr);  
\_ goto retry;  
\_ }  
\_ lock_ptr = q->lock_ptr;  
\_ if (lock_ptr != 0) {  
/* FALSE now */  
/* thread B can go on now */
```

=== end of scenario ===

So, there are two dummy and heavy task-switchings due to the

[PATCH] 2.6.16 – futex: small optimization (?)

"if (lock_ptr != 0)" statement still true the first time in unqueue_me where it should not.

Here is the patch I would like to propose, for comments.

Signed-off-by: Pierre Peiffer <Pierre.Peiffer@xxxxxxxx>

```
---
diff -uprN linux-2.6.16.ori/kernel/futex.c linux-2.6.16/kernel/futex.c
--- linux-2.6.16.ori/kernel/futex.c 2006-03-27 16:52:11.000000000 +0200
+++ linux-2.6.16/kernel/futex.c 2006-03-27 16:56:35.000000000 +0200
@@ -290,7 +290,7 @@ static int futex_wake(unsigned long uadd
struct futex_hash_bucket *bh;
struct list_head *head;
struct futex_q *this, *next;
- int ret;
+ int ret, drop_count=0;

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

@@ -305,12 +305,15 @@ static int futex_wake(unsigned long uadd
list_for_each_entry_safe(this, next, head, list) {
if (match_futex (&this->key, &key)) {
wake_futex(this);
+ drop_count++;
if (++ret >= nr_wake)
break;
}
}

spin_unlock(&bh->lock);
+ while (--drop_count >= 0)
+ drop_key_refs(&key);
out:
up_read(&current->mm->mmap_sem);
return ret;
@@ -327,6 +330,7 @@ static int futex_wake_op(unsigned long u
struct list_head *head;
struct futex_q *this, *next;
int ret, op_ret, attempt = 0;
+ int drop_count1 = 0, drop_count2 = 0;

retryfull:
down_read(&current->mm->mmap_sem);
@@ -413,6 +417,7 @@ retry:
list_for_each_entry_safe(this, next, head, list) {
if (match_futex (&this->key, &key1)) {
wake_futex(this);
+ drop_count1++;
if (++ret >= nr_wake)
```

[PATCH] 2.6.16 – futex: small optimization (?)

[PATCH] 2.6.16 – futex: small optimization (?)

```
break;
}
@@ -425,6 +430,7 @@ retry:
list_for_each_entry_safe(this, next, head, list) {
if (match_futex (&this->key, &key2)) {
wake_futex(this);
+ drop_count2++;
if (++op_ret >= nr_wake2)
break;
}
@@ -435,6 +441,12 @@ retry:
spin_unlock(&bh1->lock);
if (bh1 != bh2)
spin_unlock(&bh2->lock);
+
+ /* drop_key_refs() must be called outside the spinlocks. */
+ while (--drop_count1 >= 0)
+ drop_key_refs(&key1);
+ while (--drop_count2 >= 0)
+ drop_key_refs(&key2);
out:
up_read(&current->mm->mmap_sem);
return ret;
@@ -506,6 +518,7 @@ static int futex_requeue(unsigned long u
continue;
if (++ret <= nr_wake) {
wake_futex(this);
+ drop_count++;
} else {
list_move_tail(&this->list, &bh2->chain);
this->lock_ptr = &bh2->lock;
@@ -586,6 +599,9 @@ static int unqueue_me(struct futex_q *q)
int ret = 0;
spinlock_t *lock_ptr;

+ if (list_empty(&q->list))
+ return ret;
+
/* In the common case we don't take the spinlock, which is nice. */
retry:
lock_ptr = q->lock_ptr;
```

--
Pierre Peiffer

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

[PATCH] 2.6.16 – futex: small optimization (?)