

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

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-03/msg09524.html>

- *From:* Eric Dumazet <dada1@xxxxxxxxxxxxxx>
 - *Date:* Tue, 28 Mar 2006 12:05:44 +0200
-

Pierre PEIFFER a écrit :

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).

Its true that futex code implies lot of context switches (kernel side but also user side).

Even if you change kernel behavior in futex_wake(), you wont change the fact that a typical pthread_cond_signal does :

- 1) lock cond var
ll_lock(cv->lock);
- 2) wake one waiter if necessary
FUTEX_WAKE(cv->wakeup_seq, 1);
- 3) unlock cond var

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

If a waiter process B has higher priority than the wake process A, then most probably, B is scheduled before A had a chance to unlock cond var (step 3))

So B will re-enter kernel (because of the contended cond var lock), and A will re-enter kernel too to futex_wake() process A again, but on cond var lock this time, not on condvar wakeup_seq futex.

Each time a thread enters futex kernel code, an expensive find_extend_vma() lookup is done, (expensive because of the read_lock but also the possible amount of vm_area_struct in mm_struct)

I wish futex code had a special implementation for PTHREAD_SCOPE_PROCESS futexes , where no vma lookups would be necessary at all. Most mutexes or condvar have a process private scope (not shared by different processes)

Eric

–

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxx

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>