

# Re: posix-cpu-timers revamp

---

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2008-03/msg08611.html>

---

- *From:* Roland McGrath <roland@xxxxxxxxxx>
  - *Date:* Sat, 22 Mar 2008 14:58:29 -0700 (PDT)
- 

I would really like to just ignore the 2-cpu scenario and just have two versions, the UP version and the n-way SMP version. It would make life, and maintenance, simpler.

Like I've said, it's only something to investigate for best performance. If the conditional code is encapsulated well, it will be simple to add another variant later and experiment with it.

Okay, I'll go back over this and make sure I got it right. It's interesting, though, that my current patch (written without this particular bit of knowledge) actually performs no differently from the existing mechanism.

Except for correctness in scenarios other than the one you are testing. :-)

Testing was performed using a heavily-modified version of the test that originally showed the problem. The test sets ITIMER\_PROF (if not run with "nohang" in the name of the executable) [...]

There are several important scenarios you did not test.

Analysis of combinations of all these variables is needed.

1. Tests with a few threads, like as many threads as CPUs or only 2x as many.
2. Tests with a process CPU timer set for a long expiration time.

i.e. a timer set, but that never goes off in your entire run.

(This is what a non-infinity RLIMIT\_CPU limit does.)

With the old code, a long enough timer and a small enough number of threads will never trigger a "rebalance".

Actually, after looking at the code again and thinking about it a bit, it appears that the `signal_struct.it_*_incr` field holds the actual interval as set by `setitimer`. Initially the `it_*_expires` field holds the expiration time as set by `setitimer`, but after the timer fires the first time that value becomes `<firing time>+it_*_incr`. In other words,

Re: posix-cpu-timers revamp

the first time it fires at the value set by setitimer() but from then on it fires at a time indicated by whatever the time was the last time the timer fired plus the value in it\_\*\_incr. This time is stored in signal\_struct.it\_\*\_expires.

That's correct. The it\_\*\_expires fields store itimerval.it\_value (the current timer) and the it\_\*\_incr fields store itimerval.it\_interval (the timer reload setting).

I guess I could be wrong about this, but it appears to be what the code is doing. If my analysis is correct, I really don't need a new field, since the old fields work just fine.

The analysis above is correct but your conclusion here is wrong. The current value of an itimer is a user feature, not just a piece of internal bookkeeping.

getitimer returns in it\_value the amount of time until the itimer fires, regardless of whether or not it will reload after it fires or with what value it will be reloaded. In a setitimer call, the it\_value sets the time at which the itimer must fire, regardless of the reload setting in it\_interval. Consider the case where it\_interval={0,0}; it\_value is still meaningful.

Your code causes any timer\_settime or timer\_delete call on a process CPU timer or any setrlimit call on RLIMIT\_CPU to suddenly change the itimer setting just as if the user had made some setitimer call that was never made or intended. That's wrong.

Thanks,  
Roland

—

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

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

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