

Re: sched_userret priority adjustment patch for sched_4bsd

Source: <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/arch/2004-09/0209.html>

From: Stephan Uphoff (ups_at_tree.com)

Date: 09/27/04

To: John Baldwin <jhb@FreeBSD.org>

Date: Mon, 27 Sep 2004 17:28:07 -0400

On Mon, 2004-09-27 at 14:43, John Baldwin wrote:

> On Monday 27 September 2004 01:20 pm, Stephan Uphoff wrote:

>> On Mon, 2004-09-27 at 10:16, John Baldwin wrote:

>>> On Saturday 25 September 2004 01:29 pm, Stephan Uphoff wrote:

>>>> When a thread is about to return to user space it resets its priority

>>>> to the user level priority.

>>>> However after lowering the permission its priority it needs to check if

>>>> its priority is still better than all other runnable threads.

>>>> This is currently not implemented.

>>>> Without the check the thread can block kernel or user threads with

>>>> better priority until a switch is forced by by an interrupt.

>>>>

>>>> The attached patch checks the relevant runqueues and threads without

>>>> slots in the same ksegrp and forces a thread switch if the currently

>>>> running thread is no longer the best thread to run after it changed its

>>>> priority.

>>>>

>>>> The patch should improve interactive response under heavy load

>>>> somewhat. It needs a lot of testing.

>>>>

>>> Perhaps the better fix is to teach the schedulers to set `TDF_NEEDRESCHED`

>>> based on on a comparison against `user_pri` rather than `td_priority` inside

>>> of `sched_add()`? Having the flag set by `sched_add()` is supposed to make

>>> this sort of check unnecessary. Even 4.x has the same bug I think as a

>>> process can make another process runnable after it's priority has been

>>> boosted by a `tsleep()` and `need_resched()` is only called based on a

>>> comparison of `p_pri`. Ah, 4.x doesn't have the bug because it caches the

>>> priority of `curproc` when it enters the kernel and compares against that.

>>> Thus, I think the correct fix is more like this:

>>>

>>> Index: `sched_4bsd.c`

>>>

=====
>>> RCS file: `/usr/cvs/src/sys/kern/sched_4bsd.c,v`

>>> retrieving revision 1.63

freebsd-arch: Re: sched_userret priority adjustment patch for sched_4bsd

```
>>> diff -u -r1.63 sched_4bsd.c
>>> --- sched_4bsd.c 11 Sep 2004 10:07:22 -0000 1.63
>>> +++ sched_4bsd.c 27 Sep 2004 14:12:03 -0000
>>> @@ -272,7 +272,7 @@
>>> {
>>>
>>> mtx_assert(&sched_lock, MA_OWNED);
>>> - if(td->td_priority < curthread->td_priority)
>>> + if(td->td_priority < curthread->td_ksegrp->kg_user_pri)
>>> curthread->td_flags |= TDF_NEEDRESCHED;
>>> }
>>>
>>> Index: sched_ule.c
>>>
=====
>>> RCS file: /usr/cvs/src/sys/kern/sched_ule.c,v
>>> retrieving revision 1.129
>>> diff -u -r1.129 sched_ule.c
>>> --- sched_ule.c 11 Sep 2004 10:07:22 -0000 1.129
>>> +++ sched_ule.c 27 Sep 2004 14:13:01 -0000
>>> @@ -723,7 +723,7 @@
>>> */
>>> pcpu = pcpu_find(cpu);
>>> td = pcpu->pc_curthread;
>>> - if(ke->ke_thread->td_priority < td->td_priority ||
>>> + if(ke->ke_thread->td_priority < td->td_ksegrp->kg_user_pri ||
>>> td == pcpu->pc_idlethread) {
>>> td->td_flags |= TDF_NEEDRESCHED;
>>> ipi_selected(1 << cpu, IPI_AST);
>>>
>>> An even better fix might be to fix td_base_pri by having it be set on
>>> kernel entry similar to how 4.x sets curpriority. The above fix should
>>> be sufficient for now, however.
>>>
>>> I don't think that this is enough since TDF_NEEDRESCHED is thread
>>> specific and not cpu specific.
>>>
>>> Hmm, it is CPU specific in 4.x. It could be changed back to being a per-cpu
>>> flag easily.
```

But this might not help :-).

Example:

Thread A is running in the kernel and is preempted by an interrupt

Thread I. Thread I wakes up thread B.

If $I \rightarrow td_ksegrp \rightarrow kg_user_pri \leq B \rightarrow td_priority$ TDF_NEEDRESCHED will not be set.

If $A \rightarrow td_priority < B \rightarrow td_priority$ thread A will run once I is finished serving interrupts.

Thread A can now leave the kernel also $A \rightarrow td_ksegrp \rightarrow kg_user_pri > B \rightarrow td_priority$ may be true.

freebsd-arch: Re: sched_userret priority adjustment patch for sched_4bsd

> > *However the thread marked with TDF_NEEDRESCHED might not be the next
> > thread leaving the kernel.*
> > *(Can't really talk about ULE since I am trying to avoid looking at
> > another shiny irresistible time sink this week ; -)*
> >
> > *I think we agree that that td_priority should be set to td_base_pri on
> > kernel entry. Since td_base_pri is changed by sleep and condvar
> > functions it should also be reset on kernel entry. (Probably from a new
> > ksegrp field). Condvar waits should currently non cause the base
> > priority to change to the current priority of the thread – otherwise
> > td_base_pri could get stuck at a really bad user priority.*
> > *(td->td_base_pri might end up being worse than
> > td->td_ksegrp->kg_user_pri when the ksegrp priority improves)*
>
> *Well, I think instead that td_base_pri should be set to td_priority on kernel
> entry (rather than the other way around). td_priority should be unchanged
> just because it enters the kernel.*

I guess we disagree here.

There are just too many resource dependencies in the kernel that can lead to priority inversion (vnode locks, disk buffer ownership, etc).

It would be nice to delay the priority boost until a thread acquires such a resource (or even trace resource dependencies and implement priority inheritance) ... but this would be a huge task.

Boosting the priority on kernel entry is easy and less error prone. I guess we had this discussion last week and we just disagree on the issue.

> *I think the sleep functions could then
> leave td_base_pri alone. (I think setting it there is wrong because
> td_base_pri is not quite the same as curpriority in 4.x.) What td_base_pri
> is really supposed to provide, btw, is the priority that the thread should go
> back to once it has unlocked a mutex and had its priority boosted while it
> held the mutex. Arguably it should just be using kg_user_pri for this, but
> then you lose priority "boosts" from tsleep(), which is why td_base_pri is
> set in msleep(). I guess what should happen is something more like this:*
>
> *kernel_entry()*
> {
> *KASSERT(td->td_priority == td->td_ksegrp->kg_user_pri);*
> *td->td_base_pri = td->td_priority;*
> }
>
> *msleep()*
> {
> *sched_prio(...);*
> *td_base_pri = td->td_priority;*
> }
>
> *The TDF_NEEDRESCHED checks should be using kg_user_pri as in my previous
> e-mail. Also, in sched_prio(), if our priority is ever raised (numerically,*

freebsd-arch: Re: sched_userret priority adjustment patch for sched_4bsd

- > *logically less important*), we should set *TDF_NEEDRESCHED* since we may need to
- > *switch* (4.x does this in *maybe_needresched()*). Then, *TDF_NEEDRESCHED* could
- > *become a per-cpu flag* and have it not be cleared in *mi_switch()* but be
- > *cleared only in userret()*. Hmm, I think all of the *TDF_NEEDRESCHED* handling
- > *actually beings in sched_userret()* btw.

Wouldn't this lead to unnecessary round-robin switches between threads with the same priority on sched_4bsd?

Stephan

freebsd-arch@freebsd.org mailing list

<http://lists.freebsd.org/mailman/listinfo/freebsd-arch>

To unsubscribe, send any mail to "freebsd-arch-unsubscribe@freebsd.org"