

## Re: Initial 6.1 questions

---

*Source:* <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/performance/2006-06/msg00032.html>

---

- *From:* "Kip Macy" <[kip.macy@xxxxxxxxxx](mailto:kip.macy@xxxxxxxxxx)>
  - *Date:* Tue, 13 Jun 2006 20:15:42 -0700
- 

I apologize if this e-mail seems a bit disjoint, I'm quite tired from hauling stuff around today.

I'm not entirely familiar with the system as a whole – but to give a brief rundown of what I do know:

Context switches, thread prioritization, process statistics keeping, and access to a handful of other random variables are all serialized by `sched_lock`. Process creation, process exit, process scheduling (`schedcpu()` access to the `allproc_list`) are all serialized through the `allproc_lock`.

I've discovered that `schedcpu()`'s serialization needs doesn't fit in well with `sched_lock` removal in the presence of a global process list and global runqueue (I'll skip the tedious details for now). In other words, I have missing prerequisites. My current plan for this week, once I get back from Tahoe, is in a separate branch to do the following:

- replace the global process list with a per-cpu process list hung off of `pcpu` protected by a non-interrupt disabling spinlock `pcpu_proclist_lock`
- replace the global run queue with a per-cpu runqueue hung off of `pcpu` protected by non-interrupt blocking `pcpu_runq_lock`

Once I have this stable I will integrate it into my branch where I have replaced `sched_lock` with per-thread locks and re-do the current locking I have in `choosethread()` which I believe causes performance and stability problems.

At some point it may be desirable to add support for rebalancing the `pcpu` process lists to avoid `schedcpu/ps/top` having to hold the `pcpu_proclist_lock` for too long.

Why do I say "non-interrupt blocking?". Currently we have roughly a half dozen locking primitives. The two that I am familiar with are blocking and spinning mutexes. The general policy is to use blocking locks except where a lock is used in interrupts or the scheduler. It seems to me that in the scheduler interrupts only actually need to be blocked across `cpu_switch`. Spin locks obviously have to be used because a thread cannot very well context switch while its in the

Re: Initial 6.1 questions

middle of context switching – however, provided `td_critnest > 0`, there is no reason that interrupts need to be blocked. Currently `sched_lock` is acquired in `cpu_hardclock` and `statclock` – so it does need to block interrupts. There is no reason that these two functions couldn't be run in `ast()`. In my tree I set `td_flags` atomically to avoid the need to acquire locks when setting or clearing flags. All the timer interrupt really needs to do for purposes statistics etc. is set a flag in `td_flags` indicating to `ast()` that the current thread is returning from a timer interrupt so that `cpu_hardclock` and `statclock` are called.

I have more in mind, but I'd like to keep the discussion simple by focusing on the next week or two.

–Kip

On 6/13/06, Robert Watson <rwatson@xxxxxxxxxxxx> wrote:

On Tue, 13 Jun 2006, David Xu wrote:

> On Tuesday 13 June 2006 04:32, Kris Kennaway wrote:  
>> On Mon, Jun 12, 2006 at 09:08:12PM +0100, Robert Watson wrote:  
>>> On Mon, 12 Jun 2006, Scott Long wrote:  
>>>> I run a number of high-load production systems that do a lot of network  
>>>> and filesystem activity, all with HZ set to 100. It has also been shown  
>>>> in the past that certain things in the network area where not fixed to  
>>>> deal with a high HZ value, so it's possible that it's even more  
>>>> stable/reliable with an HZ value of 100.  
>>>>  
>>>> My personal opinion is that HZ should go back down to 100 in 7-CURRENT  
>>>> immediately, and only be incremented back up when/if it's proven to be  
>>>> the right thing to do. And, I say that as someone who (errantly) pushed  
>>>> for the increase to 1000 several years ago.  
>>>>  
>>> I think it's probably a good idea to do it sooner rather than later. It  
>>> may slightly negatively impact some services that rely on frequent timers  
>>> to do things like retransmit timing and the like. But I haven't done any  
>>> measurements.  
>>>  
>> As you know, but for the benefit of the list, restoring HZ=100 is often an  
>> important performance tweak on SMP systems with many CPUs because of all  
>> the `sched_lock` activity from `statclock/hardclock`, which scales with HZ and  
>> NCPUS.  
>>  
> `sched_lock` is another big bottleneck, since if you 32 CPUs, in theory you  
> have 32X context switch speed, but now it still has only 1X speed, and there  
> are code abusing `sched_lock`, the M:N bits dynamically inserts a thread into  
> thread list at context switch time, this is a bug, this causes thread list  
> in a proc has to be protected by scheduler lock, and delivering a signal to  
> process has to hold scheduler lock and find a thread, if the proc has many

Re: Initial 6.1 questions

- > threads, this will introduce long scheduler latency, a proc lock is not
- > enough to find a thread, this is a bug, there are other code abusing
- > scheduler lock which really can use its own lock.

I've added Kip Macy to the CC, who is working with a patch for Sun4v that eliminates sched\_lock. Maybe he can comment some more on this thread?

Robert N M Watson  
Computer Laboratory  
Universty of Cambridge

---

freebsd-performance@xxxxxxxxxxx mailing list  
<http://lists.freebsd.org/mailman/listinfo/freebsd-performance>

To unsubscribe, send any mail to "freebsd-performance-unsubscribe@xxxxxxxxxxx"