

Re: Special schedulers, one CPU only kernel, one only userland

Source: <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/arch/2005-08/0112.html>

From: Robert Watson (rwatson_at_FreeBSD.org)

Date: 08/10/05

Date: Wed, 10 Aug 2005 13:22:33 +0100 (BST)

To: Andre Oppermann <andre@freebsd.org>

On Wed, 10 Aug 2005, Andre Oppermann wrote:

- > *When using FreeBSD as a high performance router there are some desirable*
- > *changes to the way multiple CPUs are handled. Normally a second CPU*
- > *doesn't add much (if any) performance to routing because of locking*
- > *overhead and packets randomly being processed by the CPUs wasting cache*
- > *efficiency. On the other hand having just one CPU is not optimal in*
- > *running the routing daemon in userland. When there are large changes to*
- > *the table (eg. BGP full feed flap) userland sucks time away from the*
- > *packet forwarding in the kernel.*
- >
- > *The idea is to combine both worlds by designating CPU0 exclusively for*
- > *all kernel processing (thus avoiding the expensive mutex synchronization*
- > *and bus locking instructions) and CPU1 exclusively for all userland*
- > *processing. Whenever a userland program does a syscall the kernel CPU*
- > *will take over. When it's done, the process get run by the userland CPU*
- > *again. That way we get a very good scalability out of two CPUs for this*
- > *particular task.*
- >
- > *Hence my question to the SMP and scheduler gurus: How well does the*
- > *current SMP and scheduler architecture lend itself to this kind of*
- > *special handling? Is it just a matter of modifying (or plugging in) the*
- > *schedule or are there more involved things to consider?*

You can get a subset of this behavior by isolating processing of, say, routing events to a specific thread, and then pinning that thread to a specific CPU. In fact, a lot of our network processing is done thread-locally, so as to avoid hitting synchronized data structures — when procesisng IP packet headers, the mbuf is kept in thread-local storage for the duration, requiring no synchronization. By extending this approach, you get the reduced synchronization benefits. The other aspect is the precedence for CPU use and avoiding bad caching behavior. Pinning gets you part of the way there. Right now I believe we don't have a way to say "Don't use CPU X for userland processes". However, the kernel should preempt userland processes when it needs to. You may find you get

freebsd-arch: Re: Special schedulers, one CPU only kernel, one only userland

90% of what you're looking for by pinning the netisr and any related
ithreads (if not using polling) to a specific CPU, then having a
thread-local routing cache or store of some sort.

Ideally the behavior you describe shouldn't require a specialized
scheduler, rather, use of existing scheduler primitives.

Robert N M Watson

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"