

Re: Network Stack Locking

Source: <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/arch/2004-05/0097.html>

From: Matthew Dillon (dillon_at_apollo.backplane.com)

Date: 05/25/04

Date: Tue, 25 May 2004 11:29:45 -0700 (PDT)

To: Alan Cox <alc@cs.rice.edu>

:>:Sounds a lot like a lot of the Mach IPC optimizations, including their use
:>:of continuations during IPC to avoid a full context switch.

:>:

:>:Robert N M Watson FreeBSD Core Team, TrustedBSD Projects

:>:robert@fledge.watson.org Senior Research Scientist, McAfee Research

:>

:> Well, I like the performance aspects of a continuation mechanism, but

:> I really dislike the memory overhead. Even a minimal stack is

:> expensive when you multiply it by potentially hundreds of thousands

:> of 'blocking' entities such as PCBs.. say, a TCP output stream.

:> Because of this the overhead and cache pollution generated by the

:> continuation mechanism increases as system load increases rather

:> then decreases.

:

:When the explicit continuation mechanism was used, the thread's stack

:was freed when the thread blocked and a new stack allocated when the

:thread was restarted. Here is a URL:

:<http://citeseer.ist.psu.edu/draves91using.html>. Notice the mention of

:space reduction in the abstract. It's worth reading.

:

:Alan

Ah, now I understand. The continuation is an exit from the procedure.

This sounds very similar to an interrupt mechanism I designed about

a decade ago. Instead of saving and restoring the interrupt

context for the interrupt thread, the thread switch was special-cased

to basically create a context (by pushing a procedure call on the stack)

on switch-in and to throw it away on switch-out (which was always at

the end of the interrupt routine). I extended the same mechanism down

into 'userland' by creating a 'waitforever()' system call which basically

did nothing but wait for and dispatch signal vectors (most of the programs

were event oriented and had no main loop). The nice thing about this was

that no context had to be saved while the program was sitting in

waitforever(), or restored when the program returned from a signal handler.

This more than doubled scheduler performance (which on a 10MHz 68000 was

freebsd-arch: Re: Network Stack Locking

important).

In many ways, the continuation mechanism and the message queue mechanism appear to be nearly identical. If an explicit exit from a procedure is required to optimize the stack with the continuation mechanism, then that isn't much different than moving the message to an event queue and returning to the message processing loop. Neither case allows you to save stack context or to save the current procedural stacking level, and both mechanisms allow you to reuse your current stack to handle multiple messages/continuations.

-Matt
Matthew Dillon
<dillon@backplane.com>

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"