

Re: Comments on the KSE option

Source: <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/current/2006-10/msg00529.html>

- *From:* Paul Allen <nospam@xxxxxxxxxxxxxxxxxxx>
 - *Date:* Fri, 27 Oct 2006 16:44:29 -0700
-

From Julian Elischer <julian@xxxxxxxxxxxxx>, Fri, Oct 27, 2006 at 04:02:43PM -0700:

In machines that are dedicated to a single task the admin is welcome to let that task hog as much as possible. When it is supposed to be shared, that is probably NOT the expected behaviour.

When one admin controls the machine, he usually picks the number of threads per task to match the work-load. He also a good opportunity to adjust the relative priorities of the tasks.

Still calling such a machine a single task system is too restrictive and presumptive.

Threads are either busy (have work to do, in which case the kernel shouldn't be making value judgements except by way of priority) or they are sleeping in which case the point is moot.

exactly, but often you get bursts of work where may many threads come to life. Does that mean everything else should stop?

Yes, precisely because the work must be done some how, some time. Now there are certain applications where latency is more critical. i.e., interactive jobs—for which the 4BSD scheduler already detects and gives a boost.

Otherwise latency is something meant to be managed via user set priority levels. What business does the kernel have second guessing that?

Where I am from, users are expected to nice long running computational tasks on their own. When they do not, they often discover that a sysop has done it for them.

Preventing users from interfering with each other on a multiuser system is a problem for rlimits to solve.

Re: Comments on the KSE option

What would you put in rlimit to solve this problem? Be specific!
Please send diffs and code outlines.

Except that it is not I who is concerned about the fairness of multiuser systems but yourself... (no offense intended)

On a single user-system, having an imbalance of consumer/producer threads is a configuration problem which again the safety net necessary is an rlimits mechanism that will allow the machine to be saved before it falls over.

On a single user system this can all be disabled anyhow..
which is why I suggest that the KSE option recently added be broken into a M:N option and a FAIRNESS option.

Actually, I agree that this makes a good amount of sense. Esp as I am under the impression that part of KSE is about ensuring proper signal delivery. No doubt there is no reason to break that just because "fairness" might not make sense.

The 1000 versus 1 is still some sort of strange academic fairness fetish. If the process with one thread is relatively (per thread) more valuable that should be reflected in the scheduling priorities and implemented in the scheduler using a mechanism similar to WFQ. Again though: this is priorities not thread grouping per process.

This is a question for the project as a whole to decide.
As I said.. "do we want 'fareness' or not?"
If not then a lot of simplification can occur. But we need to be clear on the fact that when this is ripped out it will be a one way street.
It will be hard to put back once evolution has moved the code a bit.

Fair enough.

Irrespective of that, the number of threads is usually set to match the workload and its importance.

But that depends if you are the only user, or a student doing a project on a shared machine.

It is true there are hogs in the world; such as the people who create hundreds of screen sessions that they forget to shutdown. Nonetheless, the prevailing assumption is though there are finite computational resources on a shared machine, the offered-load must be processed eventually therefore

Re: Comments on the KSE option

two much discrimination overhead is a waste of time.

When such a community resource becomes abused—whether because of liberal intentions abused or an absence of desired limit knobs—deus ex machina (the sysop) is usually needed to sort the wheat from the chaff.

Perhaps I've come down on your fairness too hard, but in my line of work—networking, I often encounter the idea that the network ought to enforce fairness on flows—above and beyond the priority markers already available. i.e., by identifying and discriminating between different tcp connections—as if whether an offered load is split into one-flow or two-flows or n-flows should be a determinate in the disposition thereof.

I rather firmly believe that all traffic with a common CoS should be treated identically regardless of whether that traffic is predominated by some flows over others.

I don't see why kernel scheduling should be different.

Anyways, "the project" is probably too well versed in my opinion now.

Thanks for listening,

Paul

freebsd-current@xxxxxxxxxxx mailing list

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

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