Subject: Re: [RFC] kernel/pid.c pid allocation wierdness
Posted by William Lee Irwin III on Fri, 16 Mar 2007 19:46:11 GMT
View Forum Message <> Reply to Message

William Lee Irwin III <wli@holomorphy.com> writes:
>> I'd not mind something better than a hashtable. The fib tree may make
>> more sense than anticipated. It's truly better to switch data structures
>> completely than fiddle with e.g. hashtable sizes. However, bear in mind
>> the degenerate space behavior of radix trees in sparse contexts.

On Fri, Mar 16, 2007 at 07:04:28AM -0600, Eric W. Biederman wrote:
> Grr. s/patricia tree/fib tree/. We use that in the networking for
> the forwarding information base and I got mis-remembered it. Anyway
> the interesting thing with the binary version of radix tree is that
> path compression is well defined. Path compression when you have
> multi-way branching is much more difficult.

Path compression isn't a big deal for multiway branching. I've usually
done it by aligning nodes and or'ing the number of levels to skip into
the lower bits of the pointer.

On Fri, Mar 16, 2007 at 07:04:28AM -0600, Eric W. Biederman wrote:
> Sure. One of the reasons to be careful with switching data
> structures. Currently the hash tables typically operate at 10:1
> unsued:used entries. 4096 entries and 100 processes.

That would be 40:1, which is "worse" in some senses. That's not
going to fly well when pid namespaces proliferate.

On Fri, Mar 16, 2007 at 07:04:28AM -0600, Eric W. Biederman wrote:
> The current work actually focuses on changing the code so we reduce
> the total number of hash table looks ups, but persistently storing
> struct pid pointers instead of storing a pid_t. This has a lot
> of benefits when it comes to implementing a pid namespace but the
> secondary performance benefit is nice as well.

I can't quite make out what you mean by all that. struct pid is already
what's in the hashtable.

On Fri, Mar 16, 2007 at 07:04:28AM -0600, Eric W. Biederman wrote:
> Although my preliminary concern was the increase in typical list
> traversal length during lookup. The current hash table typically does
> not have collisions so normally there is nothing to traverse.

Define "normally;" 10000 threads and/or processes can be standard for

some affairs.

On Fri, Mar 16, 2007 at 07:04:28AM -0600, Eric W. Biederman wrote:
> Meanwhile an rcu tree would tend towards 2-3 levels which would double
> or triple our number of cache line misses on lookup and thus reduce
> performance in the normal case.

It depends on the sort of tree used. For B+ it depends on branching factors. For hash tries (not previously discussed) with path compression new levels would only be created when the maximum node size is exceeded. For radix trees (I'm thinking hand-coded) with path compression it depends on dispersion especially above the pseudo-levels used for the lowest bits.

William Lee Irwin III <wli@holomorphy.com> writes:
>> RCU'ing radix trees is trendy but the current implementation needs a
>> spinlock where typically radix trees do not need them for RCU. I'm
>> talking this over with others interested in lockless radix tree
>> algorithms for reasons other than concurrency and/or parallelism.

On Fri, Mar 16, 2007 at 07:04:28AM -0600, Eric W. Biederman wrote:
> Sure. I was thinking of the general class of data structures not the
> existing kernel one. The multi-way branching and lack of comparisons
> looks interesting as a hash table replacement. In a lot of cases
> storing hash values in a radix tree seems a very interesting
> alternative to a traditional hash table (and in that case the keys
> are randomly distributed and can easily be made dense). For pids
> hashing the values first seems like a waste, and

Comparisons vs. no comparisons is less interesting than cachelines touched. B+ would either make RCU inconvenient or want comparisons by dereferencing, which raises the number of cachelines touched.

William Lee Irwin III <wli@holomorphy.com> writes:
>> I'd say you already have enough evidence to motivate a change of data
>> structure.

On Fri, Mar 16, 2007 at 07:04:28AM -0600, Eric W. Biederman wrote:
> I have enough to know that a better data structure could improve
> things. Getting something that is clearly better than what
> we have now is difficult. Plus I have a lot of other patches to
> coordinate. I don't think the current data structure is going to
> fall over before I get the pid namespace implemented so that is
> my current priority.

I'll look into the data structure code, then.

William Lee Irwin III <wli@holomorphy.com> writes:
>> Basically all that's needed for radix trees' space behavior to get
>> fixed up is proper path compression as opposed to the ->depth hack.
>> Done properly it also eliminates the need for a spinlock around the
>> whole radix tree for RCU.

On Fri, Mar 16, 2007 at 07:04:28AM -0600, Eric W. Biederman wrote:
> Yes. Although I have some doubts about path compression. A B+ tree
> would be a hair more expensive (as you have to binary search the keys
> before descending) and that places a greater emphasis on all of the
> keys fitting in one cache line. Still both data structures are
> interesting.

Path compression is no big deal and resolves the radix tree space
issues to my satisfaction. It's actuallly already in lib/radix-tree.c
but in an ineffective form. It's more to do with numbers of cachelines
touched. Having to fetch pid numbers from struct pid's is not going to
be very good on that front.

On Fri, Mar 16, 2007 at 07:04:28AM -0600, Eric W. Biederman wrote:
> My secondary use is that I need something for which I can do a full
> traversal with for use in readdir. If we don't have a total ordering
> that is easy and safe to store in the file offset field readdir can
> loose entries. Currently I use the pid value itself for this.
> Simply limiting tree height in the case of pids which are relatively
> dense is likely to be enough.

Manfred's readdir code should've dealt with that at least partially.
B+ trees also resolve that quite well, despite their other issues.
Sacrificing fully lockless RCU on B+ trees and wrapping writes with
spinlocks should allow pid numbers to be stored alongside pointers
in leaf nodes at the further cost of a branching factor reduction.
TLB will at least be conserved even with a degraded branching factor.

Anyway, I'm loath to use lib/radix-tree.c but a different radix tree
implementation I could run with. I've gone over some of the other
alternatives. Give me an idea of where you want me to go with the data
structure selection and I can sweep that up for you. I'm not attached
to any particular one, though I am repelled from one in particular
(which I can do anyway even if only for comparison purposes, though if
it happens to be best I'll concede and let it through anyway).

I can also defer the data structure switch to you if you really want
to reserve that for yourself.

-- wli

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers