Subject: Re: [RFC] kernel/pid.c pid allocation wierdness
Posted by ebiederm on Fri, 16 Mar 2007 13:04:28 GMT
View Forum Message <> Reply to Message

William Lee Irwin III <wli@holomorphy.com> writes:

> William Lee Irwin III <wli@holomorphy.com> writes:
>>> Radix trees' space behavior is extremely poor in sparsely-populated
>>> index spaces. There is no way they would save space or even come close
>>> to the current space footprint.
>
> On Wed, Mar 14, 2007 at 10:54:07AM -0600, Eric W. Biederman wrote:
>> Possibly. We aren't that sparsely populated when it comes to pids.
>> Hash tables aren't good at saving space either, and when they are space
>> efficient they are on the edge of long hash chains so they are on
>> the edge of performance problems. There is at least one variant of the
>> fib tree that is as space efficient as any binary tree but works by
>> looking at bits. Not that I think that one makes much sense.
>
> 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.

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.

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.

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.

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.

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.

> William Lee Irwin III <wli@holomorphy.com> writes:
>>> Lock contention here would be a severe functional regression (note
>>> "functional," not "performance;" the lock contention surrounding these
>>> affairs takes down systems vs. mere slowdown nonsense), so it would
>>> necessarily depend on lockless radix tree code for correctness.
>
> On Wed, Mar 14, 2007 at 10:54:07AM -0600, Eric W. Biederman wrote:
>> I don't know about the existing in kernel implementations but there is no
>> reason we could not have an rcu protected radix tree. At which point the
>> challenges are about the same but we have indexes that would help us find
>> the next free bit, which could reduce cpu time.
>
> 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.

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

> On Wed, Mar 14, 2007 at 10:54:07AM -0600, Eric W. Biederman wrote:
>> The current pid implementation does not scale to larger process counts
>> particularly well. The hash table size is fixed so we get a lot of hash
>> collisions etc. Several other things go wonky as well. The one time
>> I actually had 64K pids on a system (most of them zombies) it was not
>> a very pleasant situation.
>
> If you've got a microbenchmark that would be convenient for me to use
> while addressing it, unless you'd prefer to take it on yourself.
> Otherwise I can always write one myself.

I don't. I just have some observations from tracking a bug where we
were creating unreapable zombies, and so the reproducer despite trying
to reap it's zombies created a lot of zombies. I recall that ps was
noticeably slow. Beyond that I don't have good data.

> On Wed, Mar 14, 2007 at 10:54:07AM -0600, Eric W. Biederman wrote:
>> It isn't common that we push the pid count up, and with the normal pid
>> counts the current data structures seem very well suited to the
>> problem. I have been looking at the data structures though in case it
>> ever changes because the current good behavior seems quite fragile.
>> Not that I am advocating changing anything yet, but I'm thinking about

>> it so when we do come to the point where it matters we can make a
>> reasonable change.
>
> I'd say you already have enough evidence to motivate a change of data
> structure.

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.

> Tree hashing (e.g. using balanced search trees in collision
> chains) is generally good for eliminating straight-line issues of this
> form but the available in-kernel tree structures don't do so well in
> concurrent/parallel contexts and the utility of hashing becomes somewhat
> questionable afterward given the stringent limits kernel environments
> impose on pid spaces. I favor Peter Zijlstra's B+ trees once a few
> relatively minor issues are addressed on account of good behavior in
> sparse keyspaces, though cleanups (not stylistic) of radix trees' space
> behavior may yet render them suitable, and may also be more popular to
> pursue.
>
> 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.

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.

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.

> William Lee Irwin III <wli@holomorphy.com> writes:
>>> The comment block describing the hashtable locking is stale and should
>>> have been updated in tandem with the RCU changes.
>
> On Wed, Mar 14, 2007 at 10:54:07AM -0600, Eric W. Biederman wrote:

>> Feel free to submit the patch. I didn't make the RCU changes just took
>> advantage of them.
>
> I needed to note that because it and my description were in direct
> conflict. I'd rather leave it for a kernel janitor or someone who needs
> to get patches under their belt to sweep up as I've got enough laurels
> to rest on and other areas where I can get large amounts of code in
> easily enough, provided I get my act together on various fronts.

Make sense. I still haven't seen that comment yet...
Quite probably because I have a major blind spot for comments
describing how the code works unless they are higher level comments.
I just read the code. I think I only resort to reading the comments
only if I am confused as to what the code is doing or why it is doing
what it is doing.

Eric

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers