

Hello, Michal.

On Thu, Sep 27, 2012 at 02:08:06PM +0200, Michal Hocko wrote:

> Yes, because we have many users (basically almost all) who care only
> about the user memory because that's what occupies the vast majority of
> the memory. They usually want to isolate workload which would disrupt
> the global memory otherwise (e.g. backup process vs. database). You
> really do not want to pay an additional overhead for kmem accounting
> here.

I'm not too convinced. First of all, the overhead added by kmemcg isn't big. The hot path overhead is quite minimal - it doesn't do much more than indirecting one more time. In terms of memory usage, it sure could lead to a bit more fragmentation but even if it gets to several megs per cgroup, I don't think that's something excessive. So, there is overhead but I don't believe it to be prohibitive.

> > So your question for global vs local switch (that again, doesn't
> > exist; only a local *limit* exists) should really be posed in the
> > following way: "Can two different use cases with different needs be
> > hosted in the same box?"

>
> I think this is a good and a relevant question. I think this boils down
> to whether you want to have trusted and untrusted workloads at the same
> machine.
> Trusted loads usually only need user memory accounting because kmem
> consumption should be really negligible (unless kernel is doing
> something really stupid and no kmem limit will help here).
> On the other hand, untrusted workloads can do nasty things that
> administrator has hard time to mitigate and setting a kmem limit can
> help significantly.

>
> IMHO such a different loads exist on a single machine quite often (Web
> server and a back up process as the most simplistic one). The per
> hierarchy accounting, therefore, sounds like a good idea without too
> much added complexity (actually the only added complexity is in the
> proper kmem.limit_in_bytes handling which is a single place).

The distinction between "trusted" and "untrusted" is something artificially created due to the assumed deficiency of kmemcg implementation. Making things like this visible to userland is a bad idea because it locks us into a place where we can't or don't need to improve the said deficiencies and end up pushing the difficult problems to somewhere else where it will likely be implemented in a

shabbier way. There sure are cases when such approach simply cannot be avoided, but I really don't think that's the case here - the overhead already seems to be at an acceptable level and we're not taking away the escape switch.

This is userland visible API. We better err on the side of being conservative than going overboard with flexibility. Even if we eventually need to make this switching fully hierarchical, we really should be doing,

1. Implement simple global switching and look for problem cases.
2. Analyze them and see whether the problem case can't be solved in a better, more intelligent way.
3. If the problem is something structurally inherent or reasonably too difficult to solve any other way, consider dumping the problem as config parameters to userland.

We can always expand the flexibility. Let's do the simple thing first. As an added bonus, it would enable using static_keys for accounting branches too.

Thanks.

--

tejun
