Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure
Posted by Glauber Costa on Wed, 26 Sep 2012 22:45:52 GMT
View Forum Message <> Reply to Message

On 09/27/2012 02:11 AM, Johannes Weiner wrote:
> On Thu, Sep 27, 2012 at 12:02:14AM +0400, Glauber Costa wrote:
>> On 09/26/2012 11:56 PM, Tejun Heo wrote:
>>> Hello,
>>>
>>> On Wed, Sep 26, 2012 at 11:46:37PM +0400, Glauber Costa wrote:
>>>> Besides not being part of cgroup core, and respecting very much both
>>>> cgroups' and basic sanity properties, kmem is an actual feature that
>>>> some people want, and some people don't. There is no reason to believe
>>>> that applications that want will live in the same environment with ones
>>>> that don't want.
>>>
>>> I don't know. It definitely is less crazy than .use_hierarchy but I
>>> wouldn't say it's an inherently different thing. I mean, what does it
>>> even mean to have u+k limit on one subtree and not on another branch?
>>> And we worry about things like what if parent doesn't enable it but
>>> its chlidren do.
>>>
>>
>> It is inherently different. To begin with, it actually contemplates two
>> use cases. It is not a work around.
>>
>> The meaning is also very well defined. The meaning of having this
>> enabled in one subtree and not in other is: Subtree A wants to track
>> kernel memory. Subtree B does not. It's that, and never more than that.
>> There is no maybes and no buts, no magic knobs that makes it behave in a
>> crazy way.
>>
>> If a children enables it but the parent does not, this does what every
>> tree does: enable it from that point downwards.
>>
>>> This is a feature which adds complexity. If the feature is necessary
>>> and justified, sure. If not, let's please not and let's err on the
>>> side of conservativeness. We can always add it later but the other
>>> direction is much harder.
>>
>> I disagree. Having kmem tracking adds complexity. Having to cope with
>> the use case where we turn it on dynamically to cope with the "user page
>> only" use case adds complexity. But I see no significant complexity
>> being added by having it per subtree. Really.
>
> Maybe not in code, but you are adding an extra variable into the
> system. "One switch per subtree" is more complex than "one switch."
> Yes, the toggle is hidden behind setting the limit, but it's still a

> toggle. The use_hierarchy complexity comes not from the file that
> enables it, but from the resulting semantics.
>

I didn't claim the complexity was in the code. I actually think the
other way around that you do, and claim that a global switch is more
complex than a per-subtree. All properties we have so far applies to
subtrees, due to cgroup's hierarchical nature. We have no global
switches like this so far, and adding one would just add a new concept
that wasn't here.

> kmem accounting is expensive and we definitely want to allow enabling
> it separately from traditional user memory accounting. But I think
> there is no good reason to not demand an all-or-nothing answer from
> the admin; either he wants kmem tracking on a machine or not. At
> least you haven't presented a convincing case, IMO.
>
> I don't think there is strong/any demand for per-node toggles, but
> once we add this behavior, people will rely on it and expect kmem
> tracking to stay local and we are stuck with it. Adding it for the
> reason that people will use it is a self-fulfilling prophecy.

I don't think this is a compatibility only switch. Much has been said in
the past about the problem of sharing. A lot of the kernel objects are
shared by nature, this is pretty much unavoidable. The answer we have
been giving to this inquiry, is that the workloads (us) interested
in kmem accounted tend to be quite local in their file accesses (and
other kernel objects as well).

It should be obvious that not all workloads are like this, and some of
them would actually prefer to have their umem limited only.

I really don't think, and correct me if I am wrong, that the problem
lays in "is there a use case for umem?", but rather, if they should be
allowed to coexist in a box.

And honestly, it seems to me totally reasonable to avoid restricting
people to run as many workloads they think they can in the same box.

>> You have the use_hierarchy fiasco in mind, and I do understand that you
>> are raising the flag and all that.
>>
>> But think in terms of functionality: This thing here is a lot more
>> similar to swap than use_hierarchy. Would you argue that memsw should be
>> per-root ?

>
> We actually do have a per-root flag that controls accounting for swap.
>
>> The reason why it shouldn't: Some people want to limit memory
>> consumption all the way to the swap, some people don't. Same with kmem.
>
> That lies in the nature of the interface: we chose k & u+k rather than
> u & u+k, so our memory.limit_in_bytes will necessarily include kmem,
> while swap is not included there. But I really doubt that there is a
> strong case for turning on swap accounting intentionally and then
> limiting memory+swap only on certain subtrees. Where would be the
> sense in that?

It makes absolute sense. Because until I go set
memory.memsw.limit_in_bytes, my subtree is not limited, which is
precisely what kmem does.

And the use cases for that are:

1) I, application A, want to use 2G of mem, and I can never swap
2) I, application B, want to use 2G of mem, but I am fine using extra 1G
in swap.

There are plenty of workloads in both the "can swap" and "can't swap"
category around.
