

Balbir Singh wrote:

> Vaidyanathan Srinivasan wrote:

>> Balbir Singh wrote:

>>> Paul Menage wrote:

>>>> On 2/19/07, Balbir Singh <balbir@in.ibm.com> wrote:

>>>>> More worrisome is the potential for use-after-free. What prevents the  
>>>>> pointer at mm->container from referring to freed memory after we're dropped  
>>>>> the lock?

>>>>>

>>>>> The container cannot be freed unless all tasks holding references to it are  
>>>>> gone,

>>>> ... or have been moved to other containers. If you're not holding  
>>>> task->alloc\_lock or one of the container mutexes, there's nothing to  
>>>> stop the task being moved to another container, and the container  
>>>> being deleted.

>>>>

>>>> If you're in an RCU section then you can guarantee that the container  
>>>> (that you originally read from the task) and its subsystems at least  
>>>> won't be deleted while you're accessing them, but for accounting like  
>>>> this I suspect that's not enough, since you need to be adding to the  
>>>> accounting stats on the correct container. I think you'll need to hold  
>>>> mm->container\_lock for the duration of memctl\_update\_rss()

>>>>

>>>> Paul

>>>>

>>> Yes, that sounds like the correct thing to do.

>>>

>> Accounting accuracy will anyway be affected when a process is migrated  
>> while it is still allocating pages. Having a lock here does not  
>> necessarily improve the accounting accuracy. Charges from the old  
>> container would have to be moved to the new container before deletion  
>> which implies all tasks have already left the container and no  
>> mm\_struct is holding a pointer to it.

>>

>> The only condition that will break our code will be if the container  
>> pointer becomes invalid while we are updating stats. This can be  
>> prevented by RCU section as mentioned by Paul. I believe explicit  
>> lock and unlock may not provide additional benefit here.

>>

>

> Yes, if the container pointer becomes invalid, then consider the following  
> scenario

>

> 1. Use RCU, get a reference to the container

- > 2. All tasks/mm's move to newer container (and the accounting information moves)
- > 3. Container is RCU deleted
- > 4. We still charge the older container that is going to be deleted soon
- > 5. Release RCU
- > 6. RCU garbage collects (callback runs)
- >
- > We end up charging/uncharging a soon to be deleted container, that is not good.
- >
- > What did I miss?

You are right. We should go with your read/write lock method. Later we can evaluate if using an RCU and then fixing the wrong charge will work better or worse.

--Vaidy

---