Subject: Re: [ckrm-tech] [RFC][PATCH][2/4] Add RSS accounting and control
Posted by Balbir Singh on Mon, 19 Feb 2007 11:56:31 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:
> On Mon, 19 Feb 2007 16:39:33 +0530 Balbir Singh <balbir@in.ibm.com> wrote:
>
>> Andrew Morton wrote:
>>> On Mon, 19 Feb 2007 16:07:44 +0530 Balbir Singh <balbir@in.ibm.com> wrote:
>>>
>>>>> +void memctlr_mm_free(struct mm_struct *mm)
>>>>> +{
>>>>> + kfree(mm->counter);
>>>>> +}
>>>>> +
>>>>> +static inline void memctlr_mm_assign_container_direct(struct mm_struct *mm,
>>>>> + struct container *cont)
>>>>> +{
>>>>> + write_lock(&mm->container_lock);
>>>>> + mm->container = cont;
>>>>> + write_unlock(&mm->container_lock);
>>>>> +}
>>>> More weird locking here.
>>>>
>>> The container field of the mm_struct is protected by a read write spin lock.
>>> That doesn't mean anything to me.
>>>
>>> What would go wrong if the above locking was simply removed? And how does
>>> the locking prevent that fault?
>>>
>> Some pages could charged to the wrong container. Apart from that I do not
>> see anything going bad (I'll double check that).
>
> Argh. Please, think about this.
>

Sure, I will. I guess I am short circuiting my thinking process :-)

> That locking *doesn't do anything*. Except for that one situation I
> described: some other holder of the lock reads mm->container twice inside
> the lock and requires that the value be the same both times (and that sort
> of code should be converted to take a local copy, so this locking here can
> be removed).
>

Yes, that makes sense.

>>>>>> +
>>>>>> + read_lock(&mm->container_lock);
>>>>>> + cont = mm->container;
>>>>>> + read_unlock(&mm->container_lock);
>>>>>> +
>>>>>> + if (!cont)
>>>>>> + goto done;
>>>>> And here. I mean, if there was a reason for taking the lock around that
>>>>> read, then testing `cont' outside the lock just invalidated that reason.
>>>>>
>>>> We took a consistent snapshot of cont. It cannot change outside the lock,
>>>> we check the value outside. I am sure I missed something.
>>> If it cannot change outside the lock then we don't need to take the lock!
>>>
>> We took a snapshot that we thought was consistent.
>
> Consistent with what? That's a single-word read inside that lock.
>

Yes, that makes sense.

>> We check for the value
>> outside. I guess there is no harm, the worst thing that could happen
>> is wrong accounting during mm->container changes (when a task changes
>> container).
>
> If container->lock is held when a task is removed from the
> container then yes, `cont' here can refer to a container to which the task
> no longer belongs.
>
> 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, that would ensure that all mm->containers are pointing elsewhere and
never to a stale value.

I hope my short-circuited brain got this right :-)

--
Warm Regards,
Balbir Singh