Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure
Posted by Glauber Costa on Tue, 26 Jun 2012 09:17:37 GMT
View Forum Message <> Reply to Message

On 06/26/2012 01:12 PM, David Rientjes wrote:
> On Mon, 25 Jun 2012, Glauber Costa wrote:
>
>> diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
>> index 83e7ba9..22479eb 100644
>> --- a/include/linux/memcontrol.h
>> +++ b/include/linux/memcontrol.h
>> @@ -21,6 +21,7 @@
>> #define _LINUX_MEMCONTROL_H
>> #include <linux/cgroup.h>
>> #include <linux/vm_event_item.h>
>> +#include <linux/hardirq.h>
>>
>> struct mem_cgroup;
>> struct page_cgroup;
>> @@ -409,6 +410,12 @@ struct sock;
>> #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>> void sock_update_memcg(struct sock *sk);
>> void sock_release_memcg(struct sock *sk);
>> +
>> +#define mem_cgroup_kmem_on 1
>> +bool __mem_cgroup_new_kmem_page(gfp_t gfp, void *handle, int order);
>> +void __mem_cgroup_commit_kmem_page(struct page *page, void *handle, int order);
>> +void __mem_cgroup_free_kmem_page(struct page *page, int order);
>> +#define is_kmem_tracked_alloc (gfp & __GFP_KMEMCG)
>> #else
>> static inline void sock_update_memcg(struct sock *sk)
>> {
>> @@ -416,6 +423,43 @@ static inline void sock_update_memcg(struct sock *sk)
>> static inline void sock_release_memcg(struct sock *sk)
>> {
>> }
>> +
>> +#define mem_cgroup_kmem_on 0
>> +#define __mem_cgroup_new_kmem_page(a, b, c) false
>> +#define __mem_cgroup_free_kmem_page(a,b)
>> +#define __mem_cgroup_commit_kmem_page(a, b, c)
>> +#define is_kmem_tracked_alloc (false)
>> #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
>> +
>> +static __always_inline
>> +bool mem_cgroup_new_kmem_page(gfp_t gfp, void *handle, int order)
>> +{
>> + if (!mem_cgroup_kmem_on)

>> + return true;
>> + if (!is_kmem_tracked_alloc)
>> + return true;
>> + if (!current->mm)
>> + return true;
>> + if (in_interrupt())
>> + return true;
>
> You can't test for current->mm in irq context, so you need to check for
> in_interrupt() first.
 >
Right, thanks.

> Also, what prevents __mem_cgroup_new_kmem_page()
> from being called for a kthread that has called use_mm() before
> unuse_mm()?

Nothing, but I also don't see how to prevent that.
At a first glance, it seems fair to me to say that if a kernel thread
uses the mm of a process, it poses as this process for any accounting
purpose.
