Subject: Re: [PATCH v3 12/28] slab: pass memcg parameter to kmem_cache_create
Posted by Glauber Costa on Tue, 29 May 2012 15:50:39 GMT
View Forum Message <> Reply to Message

On 05/29/2012 06:27 PM, Christoph Lameter wrote:
> On Fri, 25 May 2012, Glauber Costa wrote:
>
>> index 06e4a3e..7c0cdd6 100644
>> --- a/include/linux/slab_def.h
>> +++ b/include/linux/slab_def.h
>> @@ -102,6 +102,13 @@ struct kmem_cache {
>>   */
>>   };
>>
>> +static inline void store_orig_align(struct kmem_cache *cachep, int orig_align)
>> +{
>> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>> + cachep->memcg_params.orig_align = orig_align;
>> +#endif
>> +}
>> +
>
> Why do you need to store the original alignment? Is the calculated
> alignment not enough?

I think this one can go. You are right.

>> +++ b/mm/slab.c
>> @@ -1729,6 +1729,31 @@ void __init kmem_cache_init_late(void)
>>   */
>>   }
>>
>> +static int __init memcg_slab_register_all(void)
>> +{
>> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>> + struct kmem_cache *cachep;
>> + struct cache_sizes *sizes;
>> +
>> + sizes = malloc_sizes;
>> +
>> + while (sizes->cs_size != ULONG_MAX) {
>> + if (sizes->cs_cachep)
>> +   mem_cgroup_register_cache(NULL, sizes->cs_cachep);
>> + if (sizes->cs_dmacachep)
>> +   mem_cgroup_register_cache(NULL, sizes->cs_dmacachep);
>> + sizes++;
>> + }

>> +
>> + mutex_lock(&cache_chain_mutex);
>> + list_for_each_entry(cachep,&cache_chain, next)
>> + mem_cgroup_register_cache(NULL, cachep);
>> +
>> + mutex_unlock(&cache_chain_mutex);
>> +#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
>> + return 0;
>> +}
>
> Ok this only duplicates the kmalloc arrays. Why not the others?

It does duplicate the others.

First it does a while look on the kmalloc caches, then a
list_for_each_entry in the rest. You probably missed it.

>> @@ -2331,7 +2350,7 @@ kmem_cache_create (const char *name, size_t size, size_t align,
>>   continue;
>>   }
>>
>> - if (!strcmp(pc->name, name)) {
>> + if (!memcg&& !strcmp(pc->name, name)) {
>>   printk(KERN_ERR
>>   "kmem_cache_create: duplicate cache %s\n", name);
>>   dump_stack();
>
> This implementation means that duplicate cache detection will no longer
> work within a cgroup?

For the slab, yes. For the slub, I check to see if they belong to the
same memcg.

That said, this can and should be fixed here too, thanks for spotting.

>> @@ -2543,7 +2564,12 @@ kmem_cache_create (const char *name, size_t size, size_t align,
>>   cachep->ctor = ctor;
>>   cachep->name = name;
>>
>> + if (g_cpucache_up>= FULL)
>> + mem_cgroup_register_cache(memcg, cachep);
>
> What happens if a cgroup was active during creation of slab xxy but
> then a process running in a different cgroup uses that slab to allocate
> memory? Is it charged to the first cgroup?

I don't see this situation ever happening. kmem_cache_create, when
called directly, will always create a global cache. It doesn't matter

which cgroups are or aren't active at this time or any other. We create copies per-cgroup, but we create it lazily, when someone will touch it.

At that point, which cache will be used depend on which process is using it.