
Subject: Re: [PATCH 11/23] slub: consider a memcg parameter in kmem_create_cache

Posted by [Frederic Weisbecker](#) on Tue, 24 Apr 2012 14:03:31 GMT

[View Forum Message](#) <> [Reply to Message](#)

On Fri, Apr 20, 2012 at 06:57:19PM -0300, Glauber Costa wrote:

```
> diff --git a/mm/slub.c b/mm/slub.c
> index 2652e7c..86e40cc 100644
> --- a/mm/slub.c
> +++ b/mm/slub.c
> @@ -32,6 +32,7 @@
> #include <linux/prefetch.h>
>
> #include <trace/events/kmem.h>
> +#include <linux/memcontrol.h>
>
> /*
>  * Lock order:
> @@ -3880,7 +3881,7 @@ static int slab_unmergeable(struct kmem_cache *s)
> return 0;
> }
>
> -static struct kmem_cache *find_mergeable(size_t size,
> +static struct kmem_cache *find_mergeable(struct mem_cgroup *memcg, size_t size,
> size_t align, unsigned long flags, const char *name,
> void (*ctor)(void *))
> {
> @@ -3916,21 +3917,29 @@ static struct kmem_cache *find_mergeable(size_t size,
> if (s->size - size >= sizeof(void *))
> continue;
>
> + if (memcg && s->memcg_params.memcg != memcg)
> + continue;
> +
```

This probably won't build without CONFIG_CGROUP_MEM_RES_CTLR_KMEM ?

```
> return s;
> }
> return NULL;
> }
>
> -struct kmem_cache *kmem_cache_create(const char *name, size_t size,
> - size_t align, unsigned long flags, void (*ctor)(void *))
> +struct kmem_cache *
> +kmem_cache_create_memcg(struct mem_cgroup *memcg, const char *name, size_t size,
```

Does that build without CONFIG_CGROUP_MEM_RES_CTLR ?

```

> + size_t align, unsigned long flags, void (*ctor)(void *))
> {
>   struct kmem_cache *s;
>
>   if (WARN_ON(!name))
>       return NULL;
>
>   + #ifndef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>   + WARN_ON(memcg != NULL);
>   + #endif
>   +
>   down_write(&slub_lock);
>   - s = find_mergeable(size, align, flags, name, ctor);
>   + s = find_mergeable(memcg, size, align, flags, name, ctor);
>   if (s) {
>       s->refcount++;
>       /*
>   @@ -3954,12 +3963,15 @@ struct kmem_cache *kmem_cache_create(const char *name,
>   size_t size,
>       size, align, flags, ctor)) {
>       list_add(&s->list, &slab_caches);
>       up_write(&slub_lock);
>   +   mem_cgroup_register_cache(memcg, s);

```

How do you handle when the memcg cgroup gets destroyed? Also that means only one memcg cgroup can be accounted for a given slab cache? What if that memcg cgroup has children? Hmm, perhaps this is handled in a further patch in the series, I saw a patch title with "children" inside :)

Also my knowledge on memory allocators is near zero, so I may well be asking weird questions...