
Subject: Re: [PATCH 13/23] slub: create duplicate cache
Posted by [Frederic Weisbecker](#) on Thu, 26 Apr 2012 13:10:55 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Tue, Apr 24, 2012 at 11:37:59AM -0300, Glauber Costa wrote:

> On 04/24/2012 11:18 AM, Frederic Weisbecker wrote:

> > On Sun, Apr 22, 2012 at 08:53:30PM -0300, Glauber Costa wrote:

> > > This patch provides kmem_cache_dup(), that duplicates

> > > a cache for a memcg, preserving its creation properties.

> > > Object size, alignment and flags are all respected.

> > >

> > > When a duplicate cache is created, the parent cache cannot

> > > be destructed during the child lifetime. To assure this,

> > > its reference count is increased if the cache creation

> > > succeeds.

> > >

> > > Signed-off-by: Glauber Costa<glommer@parallels.com>

> > > CC: Christoph Lameter<cl@linux.com>

> > > CC: Pekka Enberg<penberg@cs.helsinki.fi>

> > > CC: Michal Hocko<mhocko@suse.cz>

> > > CC: Kamezawa Hiroyuki<kamezawa.hiroyu@jp.fujitsu.com>

> > > CC: Johannes Weiner<hannes@cmpxchg.org>

> > > CC: Suleiman Souhlal<suleiman@google.com>

> > >---

> > include/linux/memcontrol.h | 3 +++

> > include/linux/slab.h | 3 +++

> > mm/memcontrol.c | 44 ++++++++++++++++++++++++++++++++++++++

> > mm/slub.c | 37 ++++++++++++++++++++++++++++++++++++++

> > 4 files changed, 87 insertions(+), 0 deletions(-)

> > >

> > > diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h

> > > index 99e14b9..493ecdd 100644

> > > --- a/include/linux/memcontrol.h

> > > +++ b/include/linux/memcontrol.h

> > > @@ -445,6 +445,9 @@ int memcg_css_id(struct mem_cgroup *memcg);

> > void mem_cgroup_register_cache(struct mem_cgroup *memcg,

> > struct kmem_cache *s);

> > void mem_cgroup_release_cache(struct kmem_cache *cachep);

> > +extern char *mem_cgroup_cache_name(struct mem_cgroup *memcg,

> > + struct kmem_cache *cachep);

> > +

> > #else

> > static inline void mem_cgroup_register_cache(struct mem_cgroup *memcg,

> > struct kmem_cache *s)

> > diff --git a/include/linux/slab.h b/include/linux/slab.h

> > index c7a7e05..909b508 100644

> > --- a/include/linux/slab.h

> > +++ b/include/linux/slab.h

```

>>>@@ -323,6 +323,9 @@ extern void *__kmalloc_track_caller(size_t, gfp_t, unsigned long);
>>>
>>> #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>>> #define MAX_KMEM_CACHE_TYPES 400
>>>+extern struct kmem_cache *kmem_cache_dup(struct mem_cgroup *memcg,
>>>+    struct kmem_cache *cachep);
>>>+void kmem_cache_drop_ref(struct kmem_cache *cachep);
>>> #else
>>> #define MAX_KMEM_CACHE_TYPES 0
>>> #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
>>>diff --git a/mm/memcontrol.c b/mm/memcontrol.c
>>>index 0015ed0..e881d83 100644
>>>--- a/mm/memcontrol.c
>>>+++ b/mm/memcontrol.c
>>>@@ -467,6 +467,50 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
>>> EXPORT_SYMBOL(tcp_proto_cgroup);
>>> #endif /* CONFIG_INET */
>>>
>>>+/*
>>>+ * This is to prevent races against the kmalloc cache creations.
>>>+ * Should never be used outside the core memcg code. Therefore,
>>>+ * copy it here, instead of letting it in lib/
>>>+ */
>>>+static char *kasprintf_no_account(gfp_t gfp, const char *fmt, ...)
>>>+{
>>>+ unsigned int len;
>>>+ char *p = NULL;
>>>+ va_list ap, aq;
>>>+
>>>+ va_start(ap, fmt);
>>>+ va_copy(aq, ap);
>>>+ len = vsnprintf(NULL, 0, fmt, aq);
>>>+ va_end(aq);
>>>+
>>>+ p = kmalloc_no_account(len+1, gfp);
>>>
>>>I can't seem to find kmalloc_no_account() in this patch or may be
>>>I missed it in a previous one?
>>>
>>>It is in a previous one (actually two, one for the slab, one for the
>>>slub). They are bundled in the cache creation, but I could separate
>>>it
>>>for clarity, if you prefer.

```

They seem to be the 14th and 16th patches. They should probably be before the current one for review clarity, so we define that function before it gets used. This is also good to not break bisection.

```

>
>
> >>+ if (!p)
> >>+ goto out;
> >>+
> >>+ vsnprintf(p, len+1, fmt, ap);
> >>+
> >>+out:
> >>+ va_end(ap);
> >>+ return p;
> >>+}
> >>+
> >>+char *mem_cgroup_cache_name(struct mem_cgroup *memcg, struct kmem_cache
*cachep)
> >>+{
> >>+ char *name;
> >>+ struct dentry *dentry = memcg->css.cgroup->dentry;
> >>+
> >>+ BUG_ON(dentry == NULL);
> >>+
> >>+ /* Preallocate the space for "dead" at the end */
> >>+ name = kasprintf_no_account(GFP_KERNEL, "%s(%d:%s)dead",
> >>+   cachep->name, css_id(&memcg->css), dentry->d_name.name);
> >>+
> >>+ if (name)
> >>+ /* Remove "dead" */
> >>+ name[strlen(name) - 4] = '\0';
> >
> >Why this space for "dead" ?
>
> Ok, sorry. Since I didn't include the destruction part, it got too
> easy for whoever wasn't following the last discussion on this to get
> lost - My bad. So here it is:
>
> When we destroy the memcg, some objects may still hold the cache in
> memory. It is like a reference count, in a sense, which each object
> being a reference.
>
> In typical cases, like non-shrinkable caches that has create -
> destroy patterns, the caches will go away as soon as the tasks using
> them.
>
> But in cache-like structure like the dentry cache, the objects may
> hang around until a shrinker pass takes them out. And even then,
> some of them will live on.
>
> In this case, we will display them with "dead" in the name.

```

Ok.

>

> We could hide them, but then it gets weirder because it would be
> hard to understand where is your used memory when you need to
> inspect your system.

>

> Creating another file, slabinfo_deadaches, and keeping the names,
> is also a possibility, if people think that the string append is way
> too ugly.

Ok, thanks for the explanation.
