Subject: Re: [PATCH v6 2/2] decrement static keys on real destroy time
Posted by Glauber Costa on Wed, 23 May 2012 09:16:36 GMT
View Forum Message <> Reply to Message

On 05/23/2012 02:46 AM, Andrew Morton wrote:

```
> Here, we're open-coding kinda-test_bit().  Why do that?  These flags are
> modified with set_bit() and friends, so we should read them with the
> matching test_bit()?
```

My reasoning was to be as cheap as possible, as you noted yourself two
paragraphs below.

```
> Also, these bool-returning functions will return values other than 0
> and 1.  That probably works OK and I don't know what the C standards
> and implementations do about this.  But it seems unclean and slightly
> risky to have a "bool" value of 32!  Converting these functions to use
> test_bit() fixes this - test_bit() returns only 0 or 1.
>
> test_bit() is slightly more expensive than the above.  If this is
> considered to be an issue then I guess we could continue to use this
> approach.  But I do think a code comment is needed, explaining and
> justifying the unusual decision to bypass the bitops API.  Also these
> functions should tell the truth and return an "int" type.
>
>> >
>> >  +static void disarm_static_keys(struct mem_cgroup *memcg)
>> >  +{
>> >  + disarm_sock_keys(memcg);
>> >  +}
> Why does this function exist?  Its single caller could call
> disarm_sock_keys() directly.
```

It exists to make it clear that this is the point in which static keys
should be disabled. I already have a patchset that introduces other
static keys, that should, of course, also be disabled here.

I am totally fine with calling directly disarm_sock_keys(), and then in
that series wrap it in disarm_static_keys, IOW, defer its introduction,
if that's how you prefer.

```
>
>> >    static void drain_all_stock_async(struct mem_cgroup *memcg);
>> >
>> >    static struct mem_cgroup_per_zone *
>> >  @@ -4836,6 +4854,13 @@ static void free_work(struct work_struct *work)
>> >     int size = sizeof(struct mem_cgroup);
>> >
>> >     memcg = container_of(work, struct mem_cgroup, work_freeing);
>> >  + /*
>> >  +  * We need to make sure that (at least for now), the jump label
>> >  +  * destruction code runs outside of the cgroup lock.
> This is a poor comment - it failed to tell the reader*why*  that code
> must run outside the cgroup lock.
```

Ok, will update.

```
>> >      schedule_work()
>> >  +  * will guarantee this happens. Be careful if you need to move this
>> >  +  * disarm_static_keys around
> It's a bit difficult for the reader to be careful when he isn't told
> what the risks are.
```

Ok, will update.

```
>> >  +  */
>> >  + disarm_static_keys(memcg);
>> >     if (size<  PAGE_SIZE)
>> >      kfree(memcg);
>> >     else
>> >  diff --git a/net/ipv4/tcp_memcontrol.c b/net/ipv4/tcp_memcontrol.c
>> >  index 1517037..3b8fa25 100644
>> >  --- a/net/ipv4/tcp_memcontrol.c
>> >  +++ b/net/ipv4/tcp_memcontrol.c
>> >  @@ -74,9 +74,6 @@ void tcp_destroy_cgroup(struct mem_cgroup *memcg)
>> >     percpu_counter_destroy(&tcp->tcp_sockets_allocated);
>> >
>> >     val = res_counter_read_u64(&tcp->tcp_memory_allocated, RES_LIMIT);
>> >  -
>> >  - if (val != RESOURCE_MAX)
>> >  -  static_key_slow_dec(&memcg_socket_limit_enabled);
>> >    }
>> >    EXPORT_SYMBOL(tcp_destroy_cgroup);
>> >
>> >  @@ -107,10 +104,33 @@ static int tcp_update_limit(struct mem_cgroup *memcg, u64 val)
>> >      tcp->tcp_prot_mem[i] = min_t(long, val>>  PAGE_SHIFT,
>> >         net->ipv4.sysctl_tcp_mem[i]);
>> >
>> >  - if (val == RESOURCE_MAX&&  old_lim != RESOURCE_MAX)
>> >  -  static_key_slow_dec(&memcg_socket_limit_enabled);
>> >  - else if (old_lim == RESOURCE_MAX&&  val != RESOURCE_MAX)
>> >  -  static_key_slow_inc(&memcg_socket_limit_enabled);
>> >  + if (val == RESOURCE_MAX)
>> >  +  clear_bit(MEMCG_SOCK_ACTIVE,&cg_proto->flags);
>> >  + else if (val != RESOURCE_MAX) {
>> >  +  /*
>> >  +   *  The active bit needs to be written after the static_key update.
>> >  +   *  This is what guarantees that the socket activation function
>> >  +   *  is the last one to run. See sock_update_memcg() for details,
>> >  +   *  and note that we don't mark any socket as belonging to this
>> >  +   *  memcg until that flag is up.
>> >  +   *
>> >  +   *  We need to do this, because static_keys will span multiple
>> >  +   *  sites, but we can't control their order. If we mark a socket
>> >  +   *  as accounted, but the accounting functions are not patched in
>> >  +   *  yet, we'll lose accounting.
>> >  +   *
>> >  +   *  We never race with the readers in sock_update_memcg(), because
>> >  +   *  when this value change, the code to process it is not patched in
>> >  +   *  yet.
>> >  +   *
>> >  +   *  The activated bit is used to guarantee that no two writers will
>> >  +   *  do the update in the same memcg. Without that, we can't properly
>> >  +   *  shutdown the static key.
>> >  +   */
```
> This comment needlessly overflows 80 cols and has a pointless and
> unconventional double-space indenting. I already provided a patch
> which fixes this and a few other things, but that was ignored when you
> did the v6.

Sorry, I missed it.

>
```
>> >  +  if (!test_and_set_bit(MEMCG_SOCK_ACTIVATED,&cg_proto->flags))
>> >  +   static_key_slow_inc(&memcg_socket_limit_enabled);
>> >  +  set_bit(MEMCG_SOCK_ACTIVE,&cg_proto->flags);
>> >  + }
```
> So here are suggested changes from*some* of the above discussion.
> Please consider, incorporate, retest and send us a v7?

How do you want me to do it? Should I add your patch ontop of mine,
and then another one that tweaks whatever else is left, or should I just
merge those changes into the patches I have?