

---

Subject: Re: [PATCH v4 08/14] res\_counter: return amount of charges after res\_counter\_uncharge

Posted by [Glauber Costa](#) on Wed, 10 Oct 2012 09:03:39 GMT

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

---

On 10/09/2012 07:35 PM, Michal Hocko wrote:

> On Tue 09-10-12 19:14:57, Glauber Costa wrote:

>> On 10/09/2012 07:08 PM, Michal Hocko wrote:

>>> As I have already mentioned in my previous feedback this is certainly not

>>> atomic as you the lock protects only one group in the hierarchy. How is

>>> the return value from this function supposed to be used?

>>

>> So, I tried to make that clearer in the updated changelog.

>>

>> Only the value of the base memcg (the one passed to the function) is

>> returned, and it is atomic, in the sense that it has the same semantics

>> as the atomic variables: If 2 threads uncharge 4k each from a 8 k

>> counter, a subsequent read can return 0 for both. The return value here

>> will guarantee that only one sees the drop to 0.

>>

>> This is used in the patch "kmem\_accounting lifecycle management" to be

>> sure that only one process will call mem\_cgroup\_put() in the memcg

>> structure.

>

> Yes, you are using res\_counter\_uncharge and its semantic makes sense.

> I was referring to res\_counter\_uncharge\_until (you removed that context

> from my reply) because that one can race resulting that nobody sees 0

> even though that parents get down to 0 as a result:

> A

> |

> B

> /\

> C(x) D(y)

>

> D and C uncharge everything.

>

> CPU0 CPU1

> ret += uncharge(D) [0] ret += uncharge(C) [0]

> ret += uncharge(B) [x-from C]

> ret += uncharge(B) [0]

> ret += uncharge(A) [y-from D]

> ret += uncharge(A) [0]

>

> ret == x ret == y

>

Sorry Michal, I didn't realize you were talking about res\_counter\_uncharge\_until.

I don't really need `res_counter_uncharge_until` to return anything, so I can just remove that if you prefer, keeping just the main `res_counter_uncharge`.

However, I still can't make sense of your concern.

The return value will return the value of the counter passed as a parameter to the function:

```
r = res_counter_uncharge_locked(c, val);  
if (c == counter)  
    ret = r;
```

So when you call `res_counter_uncharge_until(D, whatever, x)`, you will see zero here as a result, and when you call `res_counter_uncharge_until(D, whatever, y)` you will see 0 here as well.

A doesn't get involved with that.

---