Subject: Re: [PATCH 6/6] containers: BeanCounters over generic process containers
Posted by Pavel Emelianov on Mon, 25 Dec 2006 10:35:15 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Fri, Dec 22, 2006 at 06:14:48AM -0800, Paul Menage wrote:
>> This patch implements the BeanCounter resource control abstraction
>> over generic process containers. It contains the beancounter core
>> code, plus the numfiles resource counter. It doesn't currently contain
>> any of the memory tracking code or the code for switching beancounter
>> context in interrupts.
>
> I don't like it, it looks bloated and probably
> adds plenty of overhead (similar to the OVZ
> implementation where this seems to be taken from)

FULL BC patch w/o pages fractions accounting doesn't
add any noticeable overhead to mainstream kernel.
Pages fractions accounting will be optimized as well.
The part you're talking about is only 1/100 of the
complete patch.

> here are some comments/questions:
>
>> Currently all the beancounters resource counters are lumped into a
>> single hierarchy; ideally it would be possible for each resource
>> counter to be a separate container subsystem, allowing them to be
>> connected to different hierarchies.
>>

```
>> +static inline void bc_uncharge(struct beancounter *bc, int res_id,
>> +  unsigned long val)
>> +{
>> + unsigned long flags;
>> +
>> + spin_lock_irqsave(&bc->bc_lock, flags);
>> + bc_uncharge_locked(bc, res_id, val);
>> + spin_unlock_irqrestore(&bc->bc_lock, flags);
```

>
> why use a spinlock, when we could use atomic
> counters?

Because approach

```
if (atomic_read(&bc->barrier) > aromic_read(&bc->held))
        atomic_inc(&bc->held);
```

used in vserver accounting is not atomic ;)

Look at the comment below about charging two resources at once.

```
>
>> +int bc_charge_locked(struct beancounter *bc, int res, unsigned long val,
>> +  int strict, unsigned long flags)
>> +{
>> + struct bc_resource_parm *parm;
>> + unsigned long new_held;
>> +
>> + BUG_ON(val > BC_MAXVALUE);
>> +
>> + parm = &bc->bc_parms[res];
>> + new_held = parm->held + val;
>> +
>> + switch (strict) {
>> + case BC_LIMIT:
>> +  if (new_held > parm->limit)
>> +   break;
>> +  /* fallthrough */
>> + case BC_BARRIER:
>> +  if (new_held > parm->barrier) {
>> +   if (strict == BC_BARRIER)
>> +    break;
>> +   if (parm->held < parm->barrier &&
>> +     bc_resources[res]->bcr_barrier_hit)
>> +    bc_resources[res]->bcr_barrier_hit(bc);
>> +  }
>
> why do barrier checks with every accounting?
> there are probably a few cases where the
> checks could be independant from the accounting
```

Let's look at

```
    if (parm->held < parm->barrier &&
            bc_resources[res]->bcr_barrier_hit)
        bc_resources[res]->bcr_barrier_hit(bc);
```

code one more time.

In case of BC_LIMIT charge BC code informs resource
controller about BARRIER hit to take some actions
before hard resource shortage.

```
>> +  /* fallthrough */
>> + case BC_FORCE:
>> +  parm->held = new_held;
>> +  bc_adjust_maxheld(parm);
>
> in what cases do we want to cross the barrier?
>
>> +  return 0;
>> + default:
>> +  BUG();
>> + }
>> +
>> + if (bc_resources[res]->bcr_limit_hit)
>> +  return bc_resources[res]->bcr_limit_hit(bc, val, flags);
>> +
>> + parm->failcnt++;
>> + return -ENOMEM;
>
>> +int bc_file_charge(struct file *file)
>> +{
>> + int sev;
>> + struct beancounter *bc;
>> +
>> + task_lock(current);
>
> why do we lock current? it won't go away that
> easily, and for switching the bc, it might be
> better to use RCU or a separate lock, no?
```

This came from containers patches. BC code doesn't take
locks on fast paths.

```
>> + bc = task_bc(current);
>> + css_get_current(&bc->css);
>> + task_unlock(current);
>> +
>> + sev = (capable(CAP_SYS_ADMIN) ? BC_LIMIT : BC_BARRIER);
>> +
>> + if (bc_charge(bc, BC_NUMFILES, 1, sev)) {
>> +  css_put(&bc->css);
>> +  return -EMFILE;
>> + }
>> +
>> + file->f_bc = bc;
>> + return 0;
>> +}
>
> also note that certain limits are much more
> complicated than the (very simple) file limits
> and the code will be called at higher frequency
```

We do know it and we have "pre-charges" optimization
for frequent calls. bc->lock we've seen is used to
make two or more resources charge in only one atomic
operation, that is faster than doing atomic_inc()
for each resource as you've proposed above.

> how to handle requests like:
> try to get as 64 files or as many as available
> whatever is smaller

I promise, that if Linus will include patch that adds a syscall
to open 64 or "as many as available whatever is smaller" files
at once we'll add this functionality.

> happy xmas,
> Herbert
>
>

____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers