Subject: Re: [PATCH 6/6] containers: BeanCounters over generic process containers

Posted by dev on Mon, 25 Dec 2006 10:16:48 GMT

View Forum Message <> Reply to Message

Herbert,

>>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)
> here are some comments/questions:
Look like you have commented anything, but OpenVZ :)
sure you don't like it, cause it doesn't add racy dcache accounting and works :)
speaking about overhead: have you done a single measurement of BC code?

>>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?
1. some resources can contribute to multiple BC params, thus need to be accounted
   atomically into 2 counters.
2. spin_lock()/spin_unlock() has the same 1 lock operation on SMP on most archs
3. using atomic counters you can't get a snapshot of current usages.
4. all the performance critical resources should be handled with pre-charges,
   as it is done in TCP/IP accounting in mainstream and as it is done for
   files/kmemsize in OpenVZ.

>>+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
<<<< cause it simply doesn't worth optimizing.
its overhead is minor compared to accounting itself (atomic operations).

```
>>+  /* fallthrough */
>>+ case BC_FORCE:
>>+  parm->held = new_held;
>>+  bc_adjust_maxheld(parm);
```

>
>
> in what cases do we want to cross the barrier?
in this patchset it is not used AFAICS.
however, it was taken from the full BC patch where it is used to handle
resource denials as most gracefully as possible.

```
>>+  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?

<<<< I guess it's countainers patch issue...

```
>>+ 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

Agree with this. This patch doesn't prove that BCs can be integrated to the containers infrastructure.

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

Do you see any problems with these except for not-needed-anywhere-now? P)

Kirill