Subject: Re: [PATCH 2/6] BC: beancounters core (API)
Posted by dev on Thu, 24 Aug 2006 12:03:38 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:
[... snip ...]

>>+#define bc_charge_locked(bc, r, v, s)   (0)
>>> +#define bc_charge(bc, r, v)    (0)
>
>akpm:/home/akpm> cat t.c
>void foo(void)
>{
> (0);
>}
>akpm:/home/akpm> gcc -c -Wall t.c
>t.c: In function 'foo':
>t.c:4: warning: statement with no effect

these functions return value should always be checked (!).
i.e. it is never called like:
  ub_charge(bc, r, v);

>>+struct beancounter *beancounter_findcreate(uid_t uid, int mask)
>>+{
>>+ struct beancounter *new_bc, *bc;
>>+ unsigned long flags;
>>+ struct hlist_head *slot;
>>+ struct hlist_node *pos;
>>+
>>+ slot = &bc_hash[bc_hash_fun(uid)];
>>+ new_bc = NULL;
>>+
>>+retry:
>>+ spin_lock_irqsave(&bc_hash_lock, flags);
>>+ hlist_for_each_entry (bc, pos, slot, hash)
>>+  if (bc->bc_id == uid)
>>+   break;
>>+
>>+ if (pos != NULL) {
>>+  get_beancounter(bc);
>>+  spin_unlock_irqrestore(&bc_hash_lock, flags);
>>+
>>+  if (new_bc != NULL)
>>+   kmem_cache_free(bc_cachep, new_bc);
>>+  return bc;
>>+ }
>>+

```
>>+ if (!(mask & BC_ALLOC))
>>+  goto out_unlock;
>>+
>>+ if (new_bc != NULL)
>>+  goto out_install;
>>+
>>+ spin_unlock_irqrestore(&bc_hash_lock, flags);
>>+
>>+ new_bc = kmem_cache_alloc(bc_cachep,
>>+   mask & BC_ALLOC_ATOMIC ? GFP_ATOMIC : GFP_KERNEL);
>>+ if (new_bc == NULL)
>>+  goto out;
>>+
>>+ memcpy(new_bc, &default_beancounter, sizeof(*new_bc));
>>+ init_beancounter_struct(new_bc, uid);
>>+ goto retry;
>>+
>>+out_install:
>>+ hlist_add_head(&new_bc->hash, slot);
>>+out_unlock:
>>+ spin_unlock_irqrestore(&bc_hash_lock, flags);
>>+out:
>>+ return new_bc;
>>+}
>
>
> Can remove the global bc_hash_lock and make the locking per-hash-bucket.
it is not performance critical path IMHO.
this lock is taken on container create/change/destroy/user interfaces only.

>>+static inline void verify_held(struct beancounter *bc)
>>+{
>>+ int i;
>>+
>>+ for (i = 0; i < BC_RESOURCES; i++)
>>+  if (bc->bc_parms[i].held != 0)
>>+   bc_print_resource_warning(bc, i,
>>+     "resource is held on put", 0, 0);
>>+}
>>+
>>+void __put_beancounter(struct beancounter *bc)
>>+{
>>+ unsigned long flags;
>>+
>>+ /* equivalent to atomic_dec_and_lock_irqsave() */
>>+ local_irq_save(flags);
>>+ if (likely(!atomic_dec_and_lock(&bc->bc_refcount, &bc_hash_lock))) {
>>+  local_irq_restore(flags);
```

>>+  if (unlikely(atomic_read(&bc->bc_refcount) < 0))
>>+   printk(KERN_ERR "BC: Bad refcount: bc=%p, "
>>+     "luid=%d, ref=%d\n",
>>+     bc, bc->bc_id,
>>+     atomic_read(&bc->bc_refcount));
>>+  return;
>>+ }
>>+
>>+ BUG_ON(bc == &init_bc);
>>+ verify_held(bc);
>>+ hlist_del(&bc->hash);
>>+ spin_unlock_irqrestore(&bc_hash_lock, flags);
>>+ kmem_cache_free(bc_cachep, bc);
>>+}
>
> I wonder if it's safe and worthwhile to optimise away the local_irq_save():
>
>  if (atomic_dec_and_test(&bc->bc_refcount)) {
>   spin_lock_irqsave(&bc_hash_lock, flags);
>   if (atomic_read(&bc->bc_refcount) == 0) {
>    free it
put_beancounter can happen from IRQ context.
so we need something like atomic_dec_and_lock_irqsave()
Oleg Nesterov proposed more details.

Thanks,
Kirill