Subject: Re: [PATCH 2/6] BC: beancounters core (API)
Posted by Andrew Morton on Wed, 23 Aug 2006 16:42:02 GMT
View Forum Message <> Reply to Message

On Wed, 23 Aug 2006 15:03:07 +0400
Kirill Korotaev <dev@sw.ru> wrote:

> Core functionality and interfaces of BC:
> find/create beancounter, initialization,
> charge/uncharge of resource, core objects' declarations.
>
> Basic structures:
> bc_resource_parm - resource description
> beancounter - set of resources, id, lock
>
>
> ..
>
> +enum severity { BC_BARRIER, BC_LIMIT, BC_FORCE };

That's a bit generic-sounding. Make it bc_severity?

> +static inline void bc_adjust_held_minmax(struct beancounter *bc,
> + int resource)
> +{
> + if (bc->bc_parms[resource].maxheld < bc->bc_parms[resource].held)
> + bc->bc_parms[resource].maxheld = bc->bc_parms[resource].held;
> + if (bc->bc_parms[resource].minheld > bc->bc_parms[resource].held)
> + bc->bc_parms[resource].minheld = bc->bc_parms[resource].held;
> +}

That might be a bit big to inline.

Suggest you check that the compiler successfully CSE's the
`bc->bc_parms[resource]' evaulation. If not, create a temporary.

> +#define beancounter_findcreate(id, f) (NULL)
> +#define get_beancounter(bc) (NULL)
> +#define put_beancounter(bc) do { } while (0)
> +#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

> +#define bc_hash_fun(x) ((((x) >> 8) ^ (x)) & (BC_HASH_SIZE - 1))

Use hash_long()?

> +/*
> + * Per resource beancounting. Resources are tied to their luid.
> + * The resource structure itself is tagged both to the process and
> + * the charging resources (a socket doesn't want to have to search for
> + * things at irq time for example). Reference counters keep things in
> + * hand.
> + *
> + * The case where a user creates resource, kills all his processes and
> + * then starts new ones is correctly handled this way. The refcounters
> + * will mean the old entry is still around with resource tied to it.
> + */
> +
> +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.

```
> +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
```
