Subject: Re: [ckrm-tech] [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Balbir Singh on Sun, 20 Aug 2006 04:58:41 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:

> +/*
> + * Resource list.
> + */
> +
> +#define UB_RESOURCES 0
> +
> +struct ubparm {
> + /*
> + * A barrier over which resource allocations are failed gracefully.
> + * e.g. if the amount of consumed memory is over the barrier further
> + * sbrk() or mmap() calls fail, the existing processes are not killed.
> + */
> + unsigned long barrier;
> + /* hard resource limit */
> + unsigned long limit;
> + /* consumed resources */
> + unsigned long held;
> + /* maximum amount of consumed resources through the last period */
> + unsigned long maxheld;
> + /* minimum amount of consumed resources through the last period */
> + unsigned long minheld;
> + /* count of failed charges */
> + unsigned long failcnt;
> +};
> +

Comments to the side of the field would make it easier to read and understand
the structure. I think there are already other comments requesting for renaming
of the barrier field to hard_limit.

<snip>

> +static inline void ub_adjust_held_minmax(struct user_beancounter *ub,
> + int resource)
> +{
> + if (ub->ub_parms[resource].maxheld < ub->ub_parms[resource].held)
> + ub->ub_parms[resource].maxheld = ub->ub_parms[resource].held;
> + if (ub->ub_parms[resource].minheld > ub->ub_parms[resource].held)
> + ub->ub_parms[resource].minheld = ub->ub_parms[resource].held;
> +}

A comment here to clarify what the function does would be helpful, specially due

to the comparison above

```
if (maxheld < held)
 maxheld = held
if (minheld > held)
 minheld = held
```

<snip>

> +struct user_beancounter ub0;

How about global_ub or init_ub?

> +
> +#define ub_hash_fun(x) ((((x) >> 8) ^ (x)) & (UB_HASH_SIZE - 1))
> +#define ub_subhash_fun(p, id) ub_hash_fun((p)->ub_uid + (id) * 17)
> +

What hash properties are we looking for in the hash function? Is the hash function universal?

> +struct hlist_head ub_hash[UB_HASH_SIZE];
> +spinlock_t ub_hash_lock;
> +
> +EXPORT_SYMBOL(ub_hash);
> +EXPORT_SYMBOL(ub_hash_lock);
> +
> +/*
> + * Per user 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 user_beancounter *beancounter_findcreate(uid_t uid,
> + struct user_beancounter *p, int mask)
> +{
> + struct user_beancounter *new_ub, *ub, *tmpl_ub;
> + unsigned long flags;
> + struct hlist_head *slot;
> + struct hlist_node *pos;
> +
> + if (mask & UB_LOOKUP_SUB) {

```
> +  WARN_ON(p == NULL);
> +  tmpl_ub = &default_subbeancounter;
> +  slot = &ub_hash[ub_subhash_fun(p, uid)];
> + } else {
> +  WARN_ON(p != NULL);
> +  tmpl_ub = &default_beancounter;
> +  slot = &ub_hash[ub_hash_fun(uid)];
> + }
> + new_ub = NULL;
> +
> +retry:
> + spin_lock_irqsave(&ub_hash_lock, flags);
> + hlist_for_each_entry (ub, pos, slot, hash)
> +  if (ub->ub_uid == uid && ub->parent == p)
> +   break;
> +
> + if (pos != NULL) {
> +  get_beancounter(ub);
> +  spin_unlock_irqrestore(&ub_hash_lock, flags);
> +
> +  if (new_ub != NULL) {
> +   put_beancounter(new_ub->parent);
> +   kmem_cache_free(ub_cachep, new_ub);
> +  }
```

A comment indicative of this being a part of race handing would be useful.
Could you please consider refactoring this function if possible.

```
> +  return ub;
> + }
> +
> + if (!(mask & UB_ALLOC))
> +  goto out_unlock;
> +
> + if (new_ub != NULL)
> +  goto out_install;
> +
> + if (mask & UB_ALLOC_ATOMIC) {
> +  new_ub = kmem_cache_alloc(ub_cachep, GFP_ATOMIC);
> +  if (new_ub == NULL)
> +   goto out_unlock;
> +
> +  memcpy(new_ub, tmpl_ub, sizeof(*new_ub));
> +  init_beancounter_struct(new_ub, uid);
> +  if (p)
> +   new_ub->parent = get_beancounter(p);
> +  goto out_install;
> + }
```

```
> +
> + spin_unlock_irqrestore(&ub_hash_lock, flags);
> +
> + new_ub = kmem_cache_alloc(ub_cachep, GFP_KERNEL);
> + if (new_ub == NULL)
> +  goto out;
> +
> + memcpy(new_ub, tmpl_ub, sizeof(*new_ub));
> + init_beancounter_struct(new_ub, uid);
> + if (p)
> +  new_ub->parent = get_beancounter(p);
> + goto retry;
> +
> +out_install:
> + hlist_add_head(&new_ub->hash, slot);
> +out_unlock:
> + spin_unlock_irqrestore(&ub_hash_lock, flags);
> +out:
> + return new_ub;
> +}
> +
> +EXPORT_SYMBOL(beancounter_findcreate);
> +
```

<snip>

```
> +void __put_beancounter(struct user_beancounter *ub)
> +{
> + unsigned long flags;
> + struct user_beancounter *parent;
> +
> +again:
> + parent = ub->parent;
> + /* equevalent to atomic_dec_and_lock_irqsave() */
> + local_irq_save(flags);
> + if (likely(!atomic_dec_and_lock(&ub->ub_refcount, &ub_hash_lock))) {
> +  if (unlikely(atomic_read(&ub->ub_refcount) < 0))
> +   printk(KERN_ERR "UB: Bad ub refcount: ub=%p, "
> +     "luid=%d, ref=%d\n",
> +     ub, ub->ub_uid,
> +     atomic_read(&ub->ub_refcount));
> +  local_irq_restore(flags);
> +  return;
> + }
> +
> + if (unlikely(ub == &ub0)) {
> +  printk(KERN_ERR "Trying to put ub0\n");
> +  spin_unlock_irqrestore(&ub_hash_lock, flags);
```

> + return;
> + }
> +
> + verify_held(ub);
> + hlist_del(&ub->hash);
> + spin_unlock_irqrestore(&ub_hash_lock, flags);

Is this function called with the ub_hash_lock held()? A comment would be useful
or you could call it __put_beancounter_locked :-)

> +
> + kmem_cache_free(ub_cachep, ub);
> +
> + ub = parent;
> + if (ub != NULL)
> + goto again;

Could you please convert this to a do {} while() loop.

> +}
> +
> +EXPORT_SYMBOL(__put_beancounter);

<snip>

> +int charge_beancounter(struct user_beancounter *ub,
> + int resource, unsigned long val, enum severity strict)
> +{
> + int retval;
> + struct user_beancounter *p, *q;
> + unsigned long flags;
> +
> + retval = -EINVAL;
> + BUG_ON(val > UB_MAXVALUE);
> +
> + local_irq_save(flags);
> + for (p = ub; p != NULL; p = p->parent) {
> + spin_lock(&p->ub_lock);
> + retval = __charge_beancounter_locked(p, resource, val, strict);

Everyone in the hierarchy is charged the same amount - val?

> + spin_unlock(&p->ub_lock);
> + if (retval)
> +   goto unroll;
> + }
> +out_restore:
> + local_irq_restore(flags);

> + return retval;
> +
> +unroll:
> + for (q = ub; q != p; q = q->parent) {
> + spin_lock(&q->ub_lock);
> + __uncharge_beancounter_locked(q, resource, val);
> + spin_unlock(&q->ub_lock);
> + }
> + goto out_restore;

Too many goto's in both directions - please consider refactoring

> +void charge_beancounter_notop(struct user_beancounter *ub,
> + int resource, unsigned long val)

Whats the meaning of notop?

> +{
> + struct user_beancounter *p;
> + unsigned long flags;
> +
> + local_irq_save(flags);
> + for (p = ub; p->parent != NULL; p = p->parent) {
> + spin_lock(&p->ub_lock);
> + __charge_beancounter_locked(p, resource, val, UB_FORCE);
> + spin_unlock(&p->ub_lock);
> + }
> + local_irq_restore(flags);
> +}
> +

Could some of this code be shared with charge_beancounter to avoid duplication?

> +EXPORT_SYMBOL(charge_beancounter_notop);
> +
> +void __uncharge_beancounter_locked(struct user_beancounter *ub,
> + int resource, unsigned long val)
> +{
> + if (unlikely(ub->ub_parms[resource].held < val)) {
> + ub_print_resource_warning(ub, resource,
> +   "uncharging too much", val, 0);
> + val = ub->ub_parms[resource].held;
> + }
> + ub->ub_parms[resource].held -= val;
> + ub_adjust_held_minmax(ub, resource);
> +}
> +
> +void uncharge_beancounter(struct user_beancounter *ub,

```
> +  int resource, unsigned long val)
> +{
> + unsigned long flags;
> + struct user_beancounter *p;
> +
> + for (p = ub; p != NULL; p = p->parent) {
> +  spin_lock_irqsave(&p->ub_lock, flags);
> +  __uncharge_beancounter_locked(p, resource, val);
> +  spin_unlock_irqrestore(&p->ub_lock, flags);
> + }
> +}
> +
> +EXPORT_SYMBOL(uncharge_beancounter);
> +
> +void uncharge_beancounter_notop(struct user_beancounter *ub,
> +  int resource, unsigned long val)
> +{
> + struct user_beancounter *p;
> + unsigned long flags;
> +
> + local_irq_save(flags);
> + for (p = ub; p->parent != NULL; p = p->parent) {
> +  spin_lock(&p->ub_lock);
> +  __uncharge_beancounter_locked(p, resource, val);
> +  spin_unlock(&p->ub_lock);
> + }
> + local_irq_restore(flags);
> +}
> +
```

The code for both uncharge_beancounter() and uncharge_beancounter_notop() seems to do the same thing

```
> +
> +void __init ub_init_late(void)
> +{
> + struct user_beancounter *ub;
> +
> + ub_cachep = kmem_cache_create("user_beancounters",
> +   sizeof(struct user_beancounter),
> +   0, SLAB_HWCACHE_ALIGN, NULL, NULL);
> + if (ub_cachep == NULL)
> +  panic("Can't create ubc caches\n");
> +
> + ub = &default_beancounter;
```

Whats the relationship between ub0 and default_beancounter?

> + memset(ub, 0, sizeof(default_beancounter));
> + init_beancounter_syslimits(ub);
> + init_beancounter_struct(ub, 0);

Do we need to memset static global variables to 0?
> +
> + ub = &default_subbeancounter;
> + memset(ub, 0, sizeof(default_subbeancounter));
> + init_beancounter_nolimits(ub);
> + init_beancounter_struct(ub, 0);

Do we need to memset static global variables to 0?

> +}
>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech

--
Regards,
Balbir Singh,
Linux Technology Center,
IBM Software Labs