
Subject: Re: [ckrm-tech] [PATCH 3/7] BC: beancounters core (API)
Posted by [Chandra Seetharaman](#) on Wed, 30 Aug 2006 18:58:34 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Tue, 2006-08-29 at 18:53 +0400, Kirill Korotaev 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
>
> Signed-off-by: Pavel Emelianov <xemul@sw.ru>
> Signed-off-by: Kirill Korotaev <dev@sw.ru>
>
> ---
>
> include/bc/beancounter.h | 150 ++++++
> include/linux/types.h | 16 ++
> init/main.c | 4
> kernel/Makefile | 1
> kernel/bc/Makefile | 7 +
> kernel/bc/beancounter.c | 256 ++++++
> 6 files changed, 434 insertions(+)
>
> --- /dev/null 2006-07-18 14:52:43.075228448 +0400
> +++ ./include/bc/beancounter.h 2006-08-28 12:47:52.000000000 +0400
> @@ -0,0 +1,150 @@
> +/*
> + * include/bc/beancounter.h
> + *
> + * Copyright (C) 2006 OpenVZ. SWsoft Inc
> + *
> + */
> +
> +#ifndef _LINUX_BEANCOUNTER_H
> +#define _LINUX_BEANCOUNTER_H
> +
> +/*
> + * Resource list.
> + */
> +
> +#define BC_RESOURCES 0
> +
> +struct bc_resource_parm {
> + unsigned long barrier; /* 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 limit; /* hard resource limit */
> + unsigned long held; /* consumed resources */
> + unsigned long maxheld; /* maximum amount of consumed resources */
> + unsigned long minheld; /* minimum amount of consumed resources */
> + unsigned long failcnt; /* count of failed charges */
> +};
> +
> +/*
> + * Kernel internal part.
> + */
> +
> + #ifdef __KERNEL__
> +
> + #include <linux/spinlock.h>
> + #include <linux/list.h>
> + #include <asm/atomic.h>
> +
> + #define BC_MAXVALUE LONG_MAX
> +
> +/*
> + * Resource management structures
> + * Serialization issues:
> + * beancounter list management is protected via bc_hash_lock
> + * task pointers are set only for current task and only once
> + * refcount is managed atomically
> + * value and limit comparison and change are protected by per-bc spinlock
> + */
> +
> + struct beancounter {
> + atomic_t bc_refcount;
> + spinlock_t bc_lock;
> + bcid_t bc_id;
> + struct hlist_node hash;
> +
> + /* resources statistics and settings */
> + struct bc_resource_parm bc_parms[BC_RESOURCES];
> +};

```

Do we need the full data structure to be visible when CONFIG_BEANCOUNTERS is not defined ?

```

> +
> + enum bc_severity { BC_BARRIER, BC_LIMIT, BC_FORCE };
> +

```

```
> +/* Flags passed to beancounter_findcreate() */
> +#define BC_LOOKUP 0x00
> +#define BC_ALLOC 0x01 /* May allocate new one */
> +#define BC_ALLOC_ATOMIC 0x02 /* Allocate with GFP_ATOMIC */
```

These are required to be visible when CONFIG_BEANCOUNTERS is not defined ?

>From these definitions it is not obvious that both BC_ALLOC and BC_ALLOC_ATOMIC need to be set to allocate a beancounter atomically. More comments below (in beancounter_findcreate).

```
> +
> +#define BC_HASH_BITS 8
> +#define BC_HASH_SIZE (1 << BC_HASH_BITS)
```

Are these needed in this global file ?

```
> +
> +#ifdef CONFIG_BEANCOUNTERS
> +
> +/*
> + * This function tunes minheld and maxheld values for a given
> + * resource when held value changes
> + */
> +static inline void bc_adjust_held_minmax(struct beancounter *bc,
> + int resource)
> +{
> + struct bc_resource_parm *parm;
> +
> + parm = &bc->bc_parms[resource];
> + if (parm->maxheld < parm->held)
> + parm->maxheld = parm->held;
> + if (parm->minheld > parm->held)
> + parm->minheld = parm->held;
> +}
```

Why is function defined in this global file ? Can be moved to beancounter.c

Also, from the usages it looks like only one of the conditions would succeed (i.e when value is increased maxheld check might succeed and when value is decreased minheld check might succeed). Why not just put the single check in appropriate context ?

```
> +
> +int __must_check bc_charge_locked(struct beancounter *bc,
> + int res, unsigned long val, enum bc_severity strict);
```

Why do we need the `_locked` to be exported ?

It is needed to be exported, it needs to be `_locked_irq` since we expect the irqs to be disabled when this function is called.

```
> +int __must_check bc_charge(struct beancounter *bc,  
> + int res, unsigned long val, enum bc_severity strict);  
> +  
> +void bc_uncharge_locked(struct beancounter *bc, int res, unsigned long val);  
> +void bc_uncharge(struct beancounter *bc, int res, unsigned long val);  
> +  
> +struct beancounter *beancounter_findcreate(bcid_t id, int mask);
```

prototype do not need the parameter names, types would suffice (would save you few characters).

```
> +  
> +static inline struct beancounter *get_beancounter(struct beancounter *bc)  
> +{  
> + atomic_inc(&bc->bc_refcount);  
> + return bc;  
> +}  
> +  
> +void put_beancounter(struct beancounter *bc);  
> +  
> +void bc_init_early(void);  
> +void bc_init_late(void);  
> +void bc_init_proc(void);  
> +  
> +extern struct beancounter init_bc;  
> +extern const char *bc_rnames[];
```

Why `bc_rnames` need to be exported ? I do not see it being used outside of `beancounter.c` (leftover from `/proc` removal, I guess).

```
> +  
> +#else /* CONFIG_BEANCOUNTERS */  
> +  
> +#define beancounter_findcreate(id, f) (NULL)  
> +#define get_beancounter(bc) (NULL)  
> +#define put_beancounter(bc) do { } while (0)  
> +  
> +static inline __must_check int bc_charge_locked(struct beancounter *bc,  
> + int res, unsigned long val, enum bc_severity strict)  
> +{  
> + return 0;  
> +}  
> +
```

```

> +static inline __must_check int bc_charge(struct beancounter *bc,
> + int res, unsigned long val, enum bc_severity strict)
> +{
> + return 0;
> +}
> +
> +static inline void bc_uncharge_locked(struct beancounter *bc, int res,
> + unsigned long val)
> +{
> +}
> +
> +static inline void bc_uncharge(struct beancounter *bc, int res,
> + unsigned long val)
> +{
> +}
> +
> +#define bc_init_early() do { } while (0)
> +#define bc_init_late() do { } while (0)
> +#define bc_init_proc() do { } while (0)
> +
> +#endif /* CONFIG_BEANCOUNTERS */
> +#endif /* __KERNEL__ */
> +
> +#endif /* _LINUX_BEANCOUNTER_H */
> --- ./include/linux/types.h.bccore 2006-08-28 12:20:13.000000000 +0400
> +++ ./include/linux/types.h 2006-08-28 12:44:13.000000000 +0400
> @@ -40,6 +40,21 @@ typedef __kernel_gid32_t gid_t;
> typedef __kernel_uid16_t uid16_t;
> typedef __kernel_gid16_t gid16_t;
>
> +/*
> + * Type of beancounter id (CONFIG_BEANCOUNTERS)
> + *
> + * The ancient Unix implementations of this kind of resource management and
> + * security are built around setuid() which sets a uid value that cannot
> + * be changed again and is normally used for security purposes. That
> + * happened to be a uid_t and in simple setups at login uid = luid = euid
> + * would be the norm.
> + *
> + * Thus the Linux one happens to be a uid_t. It could be something else but
> + * for the "container per user" model whatever a container is must be able
> + * to hold all possible uid_t values. Alan Cox.

```

Is this comment block still valid/needed ?

```

> + */
> +typedef uid_t bcid_t;

```

Why do we need a typedef ? it is not opaque anyways. Can't we use a

unsigned long or unsigned int or __u32 or something else ?

```
> +
> #ifdef CONFIG_UID16
> /* This is defined by include/asm-{arch}/posix_types.h */
> typedef __kernel_old_uid_t old_uid_t;
> @@ -52,6 +67,7 @@ typedef __kernel_old_gid_t old_gid_t;
> #else
> typedef __kernel_uid_t uid_t;
> typedef __kernel_gid_t gid_t;
> +typedef __kernel_uid_t bcid_t;
> #endif /* __KERNEL__ */
>
> #if defined(__GNUC__) && !defined(__STRICT_ANSI__)
> --- ./init/main.c.bccore 2006-08-28 12:20:13.000000000 +0400
> +++ ./init/main.c 2006-08-28 12:43:34.000000000 +0400
> @@ -52,6 +52,8 @@
> #include <linux/debug_locks.h>
> #include <linux/lockdep.h>
>
> +#include <bc/beancounter.h>
> +
> #include <asm/io.h>
> #include <asm/bugs.h>
> #include <asm/setup.h>
> @@ -495,6 +497,7 @@ asmlinkage void __init start_kernel(void
> early_boot_irqs_off();
> early_init_irq_lock_class();
>
> + bc_init_early();
> /*
> * Interrupts are still disabled. Do necessary setups, then
> * enable them
> @@ -587,6 +590,7 @@ asmlinkage void __init start_kernel(void
> #endif
> fork_init(num_physpages);
> proc_caches_init();
> + bc_init_late();
> buffer_init();
> unnamed_dev_init();
> key_init();
> --- ./kernel/Makefile.bccore 2006-08-28 12:20:13.000000000 +0400
> +++ ./kernel/Makefile 2006-08-28 12:43:34.000000000 +0400
> @@ -12,6 +12,7 @@ obj-y    = sched.o fork.o exec_domain.o
>
> obj-$(CONFIG_STACKTRACE) += stacktrace.o
> obj-y += time/
> +obj-y += bc/
```

Instead of having it this way, we can have it
obj-\$(CONFIG_BEANCOUNTERS) += bc/

and the Makefile in bc will have only obj-y = beancounter.o ...

```
> obj-$(CONFIG_DEBUG_MUTEXES) += mutex-debug.o
> obj-$(CONFIG_LOCKDEP) += lockdep.o
> ifeq ($(CONFIG_PROC_FS),y)
> --- /dev/null 2006-07-18 14:52:43.075228448 +0400
> +++ ./kernel/bc/Makefile 2006-08-28 12:43:34.000000000 +0400
> @@ -0,0 +1,7 @@
> +#
> +# Beancounters (BC)
> +#
> +# Copyright (C) 2006 OpenVZ. SWsoft Inc
> +#
> +
> +obj-$(CONFIG_BEANCOUNTERS) += beancounter.o
> --- /dev/null 2006-07-18 14:52:43.075228448 +0400
> +++ ./kernel/bc/beancounter.c 2006-08-28 12:49:07.000000000 +0400
> @@ -0,0 +1,256 @@
> +/*
> + * kernel/bc/beancounter.c
> + *
> + * Copyright (C) 2006 OpenVZ. SWsoft Inc
> + * Original code by (C) 1998 Alan Cox
> + * 1998-2000 Andrey Savochkin <saw@saw.sw.com.sg>
> + */
> +
> +#include <linux/slab.h>
> +#include <linux/module.h>
> +#include <linux/hash.h>
> +
> +#include <bc/beancounter.h>
> +
> +static kmem_cache_t *bc_cachep;
> +static struct beancounter default_beancounter;
> +
> +static void init_beancounter_struct(struct beancounter *bc, bcid_t id);
> +
> +struct beancounter init_bc;
> +
> +const char *bc_rnames[] = {
> +};
> +
> +static struct hlist_head bc_hash[BC_HASH_SIZE];
> +static spinlock_t bc_hash_lock;
```

```

> + #define bc_hash_fn(bcid) (hash_long(bcid, BC_HASH_BITS))
> +
> + /*
> + * Per resource beancounting. Resources are tied to their bc id.
> + * 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(bcid_t id, int mask)
> + {
> + struct beancounter *new_bc, *bc;
> + unsigned long flags;
> + struct hlist_head *slot;
> + struct hlist_node *pos;
> +
> + slot = &bc_hash[bc_hash_fn(id)];
> + new_bc = NULL;
> +
> + retry:
> + spin_lock_irqsave(&bc_hash_lock, flags);
> + hlist_for_each_entry (bc, pos, slot, hash)
> + if (bc->bc_id == id)
> + 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 (new_bc != NULL)
> + goto out_install;
> +
> + spin_unlock_irqrestore(&bc_hash_lock, flags);
> +
> + if (!(mask & BC_ALLOC))
> + goto out;

```

If only BC_ALLOC_ATOMIC is set, the above test would always succeed, and

we will return NULL without even trying to alloc.

Either the above check has to be changed to check for both BC_ALLOC and BC_ALLOC_ATOMIC (preferred), or some comment need to be added that both the flags need to be set to allocate atomically.

```
> +
> + new_bc = kmem_cache_alloc(bc_cachep,
> + mask & BC_ALLOC_ATOMIC ? GFP_ATOMIC : GFP_KERNEL);
> + if (new_bc == NULL)
> + goto out;
> +
> + *new_bc = default_beancounter;
> + init_beancounter_struct(new_bc, id);
> + goto retry;
> +
> +out_install:
> + hlist_add_head(&new_bc->hash, slot);
> + spin_unlock_irqrestore(&bc_hash_lock, flags);
> +out:
> + return new_bc;
> +}
> +
> +void put_beancounter(struct beancounter *bc)
> +{
> + int i;
> + unsigned long flags;
> +
> + if (!atomic_dec_and_lock_irqsave(&bc->bc_refcount,
> + &bc_hash_lock, flags))
> + return;
> +
> + BUG_ON(bc == &init_bc);
> +
> + for (i = 0; i < BC_RESOURCES; i++)
> + if (bc->bc_parms[i].held != 0)
> + printk("BC: %d has %lu of %s held on put", bc->bc_id,
> + bc->bc_parms[i].held, bc_rnames[i]);
> +
> + hlist_del(&bc->hash);
> + spin_unlock_irqrestore(&bc_hash_lock, flags);
> +
> + kmem_cache_free(bc_cachep, bc);
> +}
> +
> +EXPORT_SYMBOL(put_beancounter);
```

EXPORT_SYMBOL_GPL ?

```
> +
```

```

> +/*
> + * Generic resource charging stuff
> + */
> +
> +/* called with bc->bc_lock held and interrupts disabled */
> +int bc_charge_locked(struct beancounter *bc, int resource, unsigned long val,
> + enum bc_severity strict)
> +{
> + /*
> + * bc_value <= BC_MAXVALUE, value <= BC_MAXVALUE, and only one addition
> + * at the moment is possible so an overflow is impossible.
> + */
> + bc->bc_parms[resource].held += val;

```

Instead of having teh above statement here, if we move it to be under BC_FORCE, then we can get rid of the subtraction at the end.

```

> +
> + switch (strict) {
> + case BC_BARRIER:
> + if (bc->bc_parms[resource].held >
> + bc->bc_parms[resource].barrier)
> + break;
> + /* fallthrough */
> + case BC_LIMIT:
> + if (bc->bc_parms[resource].held >
> + bc->bc_parms[resource].limit)
> + break;
> + /* fallthrough */
> + case BC_FORCE:
> + bc_adjust_held_minmax(bc, resource);

```

Only max_held might be affected by the addition, hence we can just check and set max_held here.

```

> + return 0;
> + default:
> + BUG();
> + }
> +
> + bc->bc_parms[resource].failcnt++;
> + bc->bc_parms[resource].held -= val;
> + return -ENOMEM;
> +}
> +EXPORT_SYMBOL(bc_charge_locked);

```

Does this need to be exported ?
EXPORT_SYMBOL_GPL.

```

> +
> +int bc_charge(struct beancounter *bc, int resource, unsigned long val,
> + enum bc_severity strict)
> +{
> + int retval;
> + unsigned long flags;
> +
> + BUG_ON(val > BC_MAXVALUE);
> +
> + spin_lock_irqsave(&bc->bc_lock, flags);
> + retval = bc_charge_locked(bc, resource, val, strict);
> + spin_unlock_irqrestore(&bc->bc_lock, flags);
> + return retval;
> +}
> +EXPORT_SYMBOL(bc_charge);

```

EXPORT_SYMBOL_GPL

```

> +
> +/* called with bc->bc_lock held and interrupts disabled */
> +void bc_uncharge_locked(struct beancounter *bc, int resource, unsigned long val)
> +{
> + if (unlikely(bc->bc_parms[resource].held < val)) {
> + printk("BC: overuncharging bc %d %s: val %lu, holds %lu\n",
> + bc->bc_id, bc_rnames[resource], val,
> + bc->bc_parms[resource].held);
> + val = bc->bc_parms[resource].held;
> + }
> +
> + bc->bc_parms[resource].held -= val;
> + bc_adjust_held_minmax(bc, resource);

```

only minheld might be affected here. Can just check/set minheld.

```

> +}
> +EXPORT_SYMBOL(bc_uncharge_locked);

```

This need to be exported ?

EXPORT_SYMBOL_GPL

```

> +
> +void bc_uncharge(struct beancounter *bc, int resource, unsigned long val)
> +{
> + unsigned long flags;
> +

```

BUGON needed ? (as in bc_charge)

```

> + spin_lock_irqsave(&bc->bc_lock, flags);
> + bc_uncharge_locked(bc, resource, val);

```

```
> + spin_unlock_irqrestore(&bc->bc_lock, flags);
> +}
> +EXPORT_SYMBOL(bc_uncharge);
```

EXPORT_SYMBOL_GPL

```
> +
> +/*
> + * Initialization
> + *
> + * struct beancounter contains
> + * - limits and other configuration settings
> + * - structural fields: lists, spinlocks and so on.
> + *
> + * Before these parts are initialized, the structure should be memset
> + * to 0 or copied from a known clean structure. That takes care of a lot
> + * of fields not initialized explicitly.
> + */
> +
> +static void init_beancounter_struct(struct beancounter *bc, bcid_t id)
> +{
> + atomic_set(&bc->bc_refcount, 1);
> + spin_lock_init(&bc->bc_lock);
> + bc->bc_id = id;
> +}
> +
> +static void init_beancounter_nolimits(struct beancounter *bc)
> +{
> + int k;
> +
> + for (k = 0; k < BC_RESOURCES; k++) {
> + bc->bc_parms[k].limit = BC_MAXVALUE;
> + bc->bc_parms[k].barrier = BC_MAXVALUE;
> + }
> +}
> +
> +static void init_beancounter_syslimits(struct beancounter *bc)
> +{
> + int k;
> +
> + for (k = 0; k < BC_RESOURCES; k++)
> + bc->bc_parms[k].barrier = bc->bc_parms[k].limit;
> +}
> +
> +void __init bc_init_early(void)
> +{
> + struct beancounter *bc;
> + struct hlist_head *slot;
> +
```

```

> + bc = &init_bc;
> +
> + init_beancounter_nolimits(bc);
> + init_beancounter_struct(bc, 0);
> +
> + spin_lock_init(&bc_hash_lock);
> + slot = &bc_hash[bc_hash_fn(bc->bc_id)];
> + hlist_add_head(&bc->hash, slot);
> +}
> +
> +void __init bc_init_late(void)
> +{
> + struct beancounter *bc;
> +
> + bc_cachep = kmem_cache_create("beancounters",
> + sizeof(struct beancounter),
> + 0, SLAB_HWCACHE_ALIGN|SLAB_PANIC,
> + NULL, NULL);
> +
> + bc = &default_beancounter;
> + init_beancounter_syslimits(bc);
> + init_beancounter_struct(bc, 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
--

```

```

-----
Chandra Seetharaman      | Be careful what you choose....
  - sekharan@us.ibm.com  | .....you may get it.
-----

```