Subject: [PATCH 3/13] BC: beancounters core and API
Posted by dev on Thu, 09 Nov 2006 16:49:24 GMT
View Forum Message <> Reply to Message

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 | 171 ++++++++++++++++++++++++++++++
 include/linux/types.h    |  16 ++
 init/main.c              |   3
 kernel/bc/beancounter.c  | 253 +++++++++++++++++++++++++++++++++++++++++++++++
 4 files changed, 443 insertions(+)

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/beancounter.h 2006-11-03 17:47:18.000000000 +0300
@@ -0,0 +1,171 @@
+/*
+ * include/bc/beancounter.h
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#ifndef __BEANCOUNTER_H__
+#define __BEANCOUNTER_H__
+
+enum {
+ BC_KMEMSIZE,
+ BC_PRIVVMPAGES,
+ BC_PHYSPAGES,
+ BC_NUMTASKS,
+ BC_NUMFILES,
+
+ BC_RESOURCES
+};
+
+struct bc_resource_parm {
+ unsigned long barrier;
```

```
+ unsigned long limit;
+ unsigned long held;
+ unsigned long minheld;
+ unsigned long maxheld;
+ unsigned long failcnt;
+};
+
+#ifdef __KERNEL__
+
+#include <linux/list.h>
+#include <linux/spinlock.h>
+#include <linux/init.h>
+#include <linux/configfs.h>
+#include <asm/atomic.h>
+
+#define BC_MAXVALUE ((unsigned long)LONG_MAX)
+
+enum bc_severity {
+ BC_BARRIER,
+ BC_LIMIT,
+ BC_FORCE,
+};
+
+struct beancounter;
+
+#ifdef CONFIG_BEANCOUNTERS
+
+struct bc_resource {
+ char *bcr_name;
+
+ int (*bcr_init)(struct beancounter *bc, int res);
+ int (*bcr_change)(struct beancounter *bc,
+   unsigned long new_bar, unsigned long new_lim);
+ void (*bcr_barrier_hit)(struct beancounter *bc);
+ int (*bcr_limit_hit)(struct beancounter *bc, unsigned long val,
+   unsigned long flags);
+ void (*bcr_fini)(struct beancounter *bc);
+};
+
+extern struct bc_resource *bc_resources[];
+
+struct beancounter {
+ atomic_t  bc_refcount;
+ spinlock_t  bc_lock;
+ bcid_t   bc_id;
+ struct hlist_node bc_hash;
+
+ struct bc_resource_parm bc_parms[BC_RESOURCES];
```

```
+};
+
+static inline struct beancounter *bc_get(struct beancounter *bc)
+{
+ atomic_inc(&bc->bc_refcount);
+ return bc;
+}
+
+extern void bc_put(struct beancounter *bc);
+
+#define BC_LOOKUP  0 /* Just lookup in hash
+       */
+#define BC_ALLOC  1 /* Lookup in hash and try to make
+       * new BC if no one found
+       */
+
+extern struct beancounter *bc_findcreate(bcid_t bcid, int bc_flags);
+
+static inline void bc_adjust_maxheld(struct bc_resource_parm *parm)
+{
+ if (parm->maxheld < parm->held)
+  parm->maxheld = parm->held;
+}
+
+static inline void bc_adjust_minheld(struct bc_resource_parm *parm)
+{
+ if (parm->minheld > parm->held)
+  parm->minheld = parm->held;
+}
+
+static inline void bc_init_resource(struct bc_resource_parm *parm,
+  unsigned long bar, unsigned long lim)
+{
+ parm->barrier = bar;
+ parm->limit = lim;
+ parm->held = 0;
+ parm->minheld = 0;
+ parm->maxheld = 0;
+ parm->failcnt = 0;
+}
+
+int bc_change_param(struct beancounter *bc, int res,
+  unsigned long bar, unsigned long lim);
+
+int __must_check bc_charge_locked(struct beancounter *bc, int res_id,
+  unsigned long val, int strict, unsigned long flags);
+static inline int __must_check bc_charge(struct beancounter *bc, int res_id,
+  unsigned long val, int strict)
```

```
+{
+ int ret;
+ unsigned long flags;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ ret = bc_charge_locked(bc, res_id, val, strict, flags);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+ return ret;
+}
+
+void __must_check bc_uncharge_locked(struct beancounter *bc, int res_id,
+  unsigned long val);
+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);
+}
+
+void __init bc_register_resource(int res_id, struct bc_resource *br);
+void __init bc_init_early(void);
+#else /* CONFIG_BEANCOUNTERS */
+static inline int __must_check bc_charge_locked(struct beancounter *bc, int res,
+  unsigned long val, int strict, unsigned long flags)
+{
+ return 0;
+}
+
+static inline int __must_check bc_charge(struct beancounter *bc, int res,
+  unsigned long val, int 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)
+{
+}
+
+static inline void bc_init_early(void)
```

```
+{
+}
+#endif /* CONFIG_BEANCOUNTERS */
+#endif /* __KERNEL__ */
+#endif
--- ./include/linux/types.h.bcprep 2006-11-03 17:46:25.000000000 +0300
+++ ./include/linux/types.h 2006-11-03 17:46:31.000000000 +0300
@@ -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 setluid() 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.
+ */
+typedef uid_t   bcid_t;
+
 #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 2006-11-03 17:46:10.000000000 +0300
+++ ./init/main.c 2006-11-03 17:47:18.000000000 +0300
@@ -53,6 +53,8 @@
 #include <linux/lockdep.h>
 #include <linux/pid_namespace.h>

+#include <bc/beancounter.h>
+
 #include <asm/io.h>
 #include <asm/bugs.h>
 #include <asm/setup.h>
```

```
@@ -483,6 +485,7 @@ asmlinkage void __init start_kernel(void
  char * command_line;
  extern struct kernel_param __start___param[], __stop___param[];

+ bc_init_early();
  smp_setup_processor_id();

  /*
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/beancounter.c 2006-11-03 17:47:18.000000000 +0300
@@ -0,0 +1,253 @@
+/*
+ * kernel/bc/beancounter.c
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#include <linux/sched.h>
+#include <linux/list.h>
+#include <linux/hash.h>
+#include <linux/gfp.h>
+#include <linux/slab.h>
+#include <linux/module.h>
+
+#include <bc/beancounter.h>
+
+#define BC_HASH_BITS (8)
+#define BC_HASH_SIZE (1 << BC_HASH_BITS)
+
+static int bc_dummy_init(struct beancounter *bc, int i)
+{
+ bc_init_resource(&bc->bc_parms[i], BC_MAXVALUE, BC_MAXVALUE);
+ return 0;
+}
+
+static struct bc_resource bc_dummy_res = {
+ .bcr_name = "dummy",
+ .bcr_init = bc_dummy_init,
+};
+
+struct bc_resource *bc_resources[BC_RESOURCES] = {
+ [0 ... BC_RESOURCES - 1] = &bc_dummy_res,
+};
+
+struct beancounter init_bc;
+static struct hlist_head bc_hash[BC_HASH_SIZE];
+static spinlock_t bc_hash_lock;
```

```
+static kmem_cache_t *bc_cache;
+
+static void init_beancounter_struct(struct beancounter *bc, bcid_t bcid)
+{
+ bc->bc_id = bcid;
+ spin_lock_init(&bc->bc_lock);
+ atomic_set(&bc->bc_refcount, 1);
+}
+
+struct beancounter *bc_findcreate(bcid_t bcid, int bc_flags)
+{
+ unsigned long flags;
+ struct beancounter *bc;
+ struct beancounter *new_bc;
+ struct hlist_head *head;
+ struct hlist_node *ptr;
+ int i;
+
+ head = &bc_hash[hash_long(bcid, BC_HASH_BITS)];
+ bc = NULL;
+ new_bc = NULL;
+
+retry:
+ spin_lock_irqsave(&bc_hash_lock, flags);
+ hlist_for_each (ptr, head) {
+  bc = hlist_entry(ptr, struct beancounter, bc_hash);
+  if (bc->bc_id == bcid)
+   break;
+ }
+
+ if (bc != NULL) {
+  bc_get(bc);
+  spin_unlock_irqrestore(&bc_hash_lock, flags);
+
+  if (new_bc != NULL)
+   kmem_cache_free(bc_cache, new_bc);
+  return bc;
+ }
+
+ if (new_bc != NULL) {
+  hlist_add_head(&new_bc->bc_hash, head);
+  spin_unlock_irqrestore(&bc_hash_lock, flags);
+  return new_bc;
+ }
+ spin_unlock_irqrestore(&bc_hash_lock, flags);
+
+ if (!(bc_flags & BC_ALLOC))
+  return NULL;
```

```
+
+ new_bc = kmem_cache_alloc(bc_cache, GFP_KERNEL);
+ if (new_bc == NULL)
+  return NULL;
+
+ init_beancounter_struct(new_bc, bcid);
+ for (i = 0; i < BC_RESOURCES; i++)
+  if (bc_resources[i]->bcr_init(new_bc, i))
+   goto out_unroll;
+ goto retry;
+
+out_unroll:
+ for (i--; i >= 0; i--)
+  if (bc_resources[i]->bcr_fini)
+   bc_resources[i]->bcr_fini(new_bc);
+ kmem_cache_free(bc_cache, new_bc);
+ return NULL;
+}
+
+void bc_put(struct beancounter *bc)
+{
+ int i;
+ unsigned long flags;
+
+ if (likely(!atomic_dec_and_lock_irqsave(&bc->bc_refcount,
+    &bc_hash_lock, flags)))
+  return;
+
+ hlist_del(&bc->bc_hash);
+ spin_unlock_irqrestore(&bc_hash_lock, flags);
+
+ for (i = 0; i < BC_RESOURCES; i++) {
+  if (bc_resources[i]->bcr_fini)
+   bc_resources[i]->bcr_fini(bc);
+
+  if (bc->bc_parms[i].held != 0)
+   printk(KERN_ERR "BC: Resource %s holds %lu on put\n",
+     bc_resources[i]->bcr_name,
+     bc->bc_parms[i].held);
+ }
+
+ kmem_cache_free(bc_cache, bc);
+}
+
+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);
+  }
+  /* fallthrough */
+ case BC_FORCE:
+  parm->held = new_held;
+  bc_adjust_maxheld(parm);
+  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;
+}
+
+void bc_uncharge_locked(struct beancounter *bc, int res, unsigned long val)
+{
+ struct bc_resource_parm *parm;
+
+ BUG_ON(val > BC_MAXVALUE);
+
+ parm = &bc->bc_parms[res];
+ if (unlikely(val > parm->held)) {
+  printk(KERN_ERR "BC: Uncharging too much of %s: %lu vs %lu\n",
+    bc_resources[res]->bcr_name,
+    val, parm->held);
+  val = parm->held;
```

```
+ }
+
+ parm->held -= val;
+ bc_adjust_minheld(parm);
+}
+
+int bc_change_param(struct beancounter *bc, int res,
+  unsigned long bar, unsigned long lim)
+{
+ int ret;
+
+ ret = -EINVAL;
+ if (bar > lim)
+  goto out;
+ if (bar > BC_MAXVALUE || lim > BC_MAXVALUE)
+  goto out;
+
+ ret = 0;
+ spin_lock_irq(&bc->bc_lock);
+ if (bc_resources[res]->bcr_change) {
+  ret = bc_resources[res]->bcr_change(bc, bar, lim);
+  if (ret < 0)
+   goto out_unlock;
+ }
+
+ bc->bc_parms[res].barrier = bar;
+ bc->bc_parms[res].limit = lim;
+
+out_unlock:
+ spin_unlock_irq(&bc->bc_lock);
+out:
+ return ret;
+}
+
+void __init bc_register_resource(int res_id, struct bc_resource *br)
+{
+ BUG_ON(bc_resources[res_id] != &bc_dummy_res);
+ BUG_ON(res_id >= BC_RESOURCES);
+
+ bc_resources[res_id] = br;
+}
+
+void __init bc_init_early(void)
+{
+ int i;
+
+ init_beancounter_struct(&init_bc, 0);
+
```

```
+ for (i = 0; i < BC_RESOURCES; i++) {
+  init_bc.bc_parms[i].barrier = BC_MAXVALUE;
+  init_bc.bc_parms[i].limit = BC_MAXVALUE;
+ }
+
+ spin_lock_init(&bc_hash_lock);
+ hlist_add_head(&init_bc.bc_hash, &bc_hash[hash_long(0, BC_HASH_BITS)]);
+}
+
+int __init bc_init_late(void)
+{
+ bc_cache = kmem_cache_create("beancounters",
+   sizeof(struct beancounter), 0,
+   SLAB_HWCACHE_ALIGN | SLAB_PANIC, NULL, NULL);
+ return 0;
+}
+
+__initcall(bc_init_late);
+
+EXPORT_SYMBOL(bc_resources);
+EXPORT_SYMBOL(init_bc);
+EXPORT_SYMBOL(bc_change_param);
+EXPORT_SYMBOL(bc_findcreate);
+EXPORT_SYMBOL(bc_put);
```