Subject: [PATCH] BC: resource beancounters (v3)
Posted by dev on Tue, 29 Aug 2006 14:31:56 GMT
View Forum Message <> Reply to Message

The following patch set presents base of
Resource Beancounters (BC).
BC allows to account and control consumption
of kernel resources used by group of processes.

Draft UBC description on OpenVZ wiki can be found at
http://wiki.openvz.org/UBC_parameters

The full BC patch set allows to control:
- kernel memory. All the kernel objects allocatable
on user demand should be accounted and limited
for DoS protection.
E.g. page tables, task structs, vmas etc.

- virtual memory pages. BCs allow to
limit a container to some amount of memory and
introduces 2-level OOM killer taking into account
container's consumption.
pages shared between containers are correctly
charged as fractions (tunable).

- network buffers. These includes TCP/IP rcv/snd
buffers, dgram snd buffers, unix, netlinks and
other buffers.

- minor resources accounted/limited by number:
tasks, files, flocks, ptys, siginfo, pinned dcache
mem, sockets, iptentries (for containers with
virtualized networking)

As the first step we want to propose for discussion
the most complicated parts of resource management:
kernel memory and virtual memory.
The patch set to be sent provides core for BC and
management of kernel memory only. Virtual memory
management will be sent in a couple of days.

The patches in these series are:
diff-atomic-dec-and-lock-irqsave.patch
  introduce atomic_dec_and_lock_irqsave()

diff-bc-kconfig.patch:
  Adds kernel/bc/Kconfig file with UBC options and
  includes it into arch Kconfigs

diff-bc-core.patch:
  Contains core functionality and interfaces of BC:
  find/create beancounter, initialization,
  charge/uncharge of resource, core objects' declarations.

diff-bc-task.patch:
  Contains code responsible for setting BC on task,
  it's inheriting and setting host context in interrupts.

  Task contains three beancounters:
  1. exec_bc - current context. all resources are charged
                to this beancounter.
  2. fork_bc - beancounter which is inherited by
                task's children on fork

diff-bc-syscalls.patch:
  Patch adds system calls for BC management:
  1. sys_get_bcid   - get current BC id
  2. sys_set_bcid   - changes exec_ and fork_ BCs on current
  3. sys_set_bclimit - set limits for resources consumtions
  4. sys_get_bcstat - returns limits/usages/fails for BC

diff-bc-kmem-core.patch:
  Introduces BC_KMEMSIZE resource which accounts kernel
  objects allocated by task's request.

  Objects are accounted via struct page and slab objects.
  For the latter ones each slab contains a set of pointers
  corresponding object is charged to.

  Allocation charge rules:
  1. Pages - if allocation is performed with __GFP_BC flag - page
     is charged to current's exec_bc.
  2. Slabs - kmem_cache may be created with SLAB_BC flag - in this
     case each allocation is charged. Caches used by kmalloc are
     created with SLAB_BC | SLAB_BC_NOCHARGE flags. In this case
     only __GFP_BC allocations are charged.

diff-bc-kmem-charge.patch:
  Adds SLAB_BC and __GFP_BC flags in appropriate places
  to cause charging/limiting of specified resources.

Summary of changes from v2 patch set:

* introduced atomic_dec_and_lock_irqsave()
* bc_adjust_held_minmax comment
* added __must_check for bc_*charge* funcs

* use hash_long() instead of own one
* bc/Kconfig is sourced from init/Kconfig now
* introduced bcid_t type with comment from Alan Cox
* check for barrier <= limit in sys_set_bclimit()
* removed (bc == NULL) checks
* replaced memcpy in beancounter_findcrate with assignment
* moved check 'if (mask & BC_ALLOC)' out of the lock
* removed unnecessary memset()

Summary of changes from v1 patch set:

* CONFIG_BEANCOUNTERS is 'n' by default
* fixed Kconfig includes in arches
* removed hierarchical beancounters to simplify first patchset
* removed unused 'private' pointer
* removed unused EXPORTS
* MAXVALUE redeclared as LONG_MAX
* beancounter_findcreate clarification
* renamed UBC -> BC, ub -> bc etc.
* moved BC inheritance into copy_process
* introduced reset_exec_bc() with proposed BUG_ON
* removed task_bc beancounter (not used yet, for numproc)
* fixed syscalls for sparc
* added sys_get_bcstat(): return info that was in /proc
* cond_syscall instead of #ifdefs

Many thanks to Oleg Nesterov, Alan Cox, Matt Helsley and others
for patch review and comments.

Patch set is applicable to 2.6.18-rc4-mm3

Thanks,
Kirill

Subject: [PATCH 1/7] introduce atomic_dec_and_lock_irqsave()
Posted by dev on Tue, 29 Aug 2006 14:47:35 GMT
View Forum Message <> Reply to Message

Oleg Nesterov noticed to me that the construction like
(used in beancounter patches and free_uid()):

```
local_irq_save(flags);
if (atomic_dec_and_lock(&refcnt, &lock))
 ...
```

is not that good for preemtible kernels, since with preemption
spin_lock() can schedule() to reduce latency. However, it won't schedule

if interrupts are disabled.

So this patch introduces atomic_dec_and_lock_irqsave() as a logical
counterpart to atomic_dec_and_lock().

Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
Signed-Off-By: Kirill Korotaev <dev@sw.ru>

---

 include/linux/spinlock.h | 6 ++++++
 kernel/user.c | 5 +----
 lib/dec_and_lock.c | 19 +++++++++++++++++++
 3 files changed, 26 insertions(+), 4 deletions(-)

--- ./include/linux/spinlock.h.dlirq 2006-08-28 10:17:35.000000000 +0400
+++ ./include/linux/spinlock.h 2006-08-28 11:22:37.000000000 +0400
@@ -266,6 +266,12 @@ extern int _atomic_dec_and_lock(atomic_t
 #define atomic_dec_and_lock(atomic, lock) \
   __cond_lock(lock, _atomic_dec_and_lock(atomic, lock))

+extern int _atomic_dec_and_lock_irqsave(atomic_t *atomic, spinlock_t *lock,
+ unsigned long *flagsp);
+#define atomic_dec_and_lock_irqsave(atomic, lock, flags) \
+ __cond_lock(lock, \
+   _atomic_dec_and_lock_irqsave(atomic, lock, &flags))
+
 /**
  * spin_can_lock - would spin_trylock() succeed?
  * @lock: the spinlock in question.
--- ./kernel/user.c.dlirq 2006-07-10 12:39:20.000000000 +0400
+++ ./kernel/user.c 2006-08-28 11:08:56.000000000 +0400
@@ -108,15 +108,12 @@ void free_uid(struct user_struct *up)
  if (!up)
   return;

- local_irq_save(flags);
- if (atomic_dec_and_lock(&up->__count, &uidhash_lock)) {
+ if (atomic_dec_and_lock_irqsave(&up->__count, &uidhash_lock, flags)) {
   uid_hash_remove(up);
   spin_unlock_irqrestore(&uidhash_lock, flags);
   key_put(up->uid_keyring);
   key_put(up->session_keyring);
   kmem_cache_free(uid_cachep, up);
- } else {
- local_irq_restore(flags);
  }
 }

```
--- ./lib/dec_and_lock.c.dlirq 2006-04-21 11:59:36.000000000 +0400
+++ ./lib/dec_and_lock.c 2006-08-28 11:22:08.000000000 +0400
@@ -33,3 +33,22 @@ int _atomic_dec_and_lock(atomic_t *atomi
 }

 EXPORT_SYMBOL(_atomic_dec_and_lock);
+
+/*
+ * the same, but takes the lock with _irqsave
+ */
+int _atomic_dec_and_lock_irqsave(atomic_t *atomic, spinlock_t *lock,
+  unsigned long *flagsp)
+{
+#ifdef CONFIG_SMP
+ if (atomic_add_unless(atomic, -1, 1))
+  return 0;
+#endif
+ spin_lock_irqsave(lock, *flagsp);
+ if (atomic_dec_and_test(atomic))
+  return 1;
+ spin_unlock_irqrestore(lock, *flagsp);
+ return 0;
+}
+
+EXPORT_SYMBOL(_atomic_dec_and_lock_irqsave);
```

Subject: [PATCH 2/7] BC: kconfig
Posted by dev on Tue, 29 Aug 2006 14:48:19 GMT
View Forum Message <> Reply to Message

Add kernel/bc/Kconfig file with BC options and
include it into arch Kconfigs

Signed-off-by: Pavel Emelianov <xemul@sw.ru>
Signed-off-by: Kirill Korotaev <dev@sw.ru>

```
 init/Kconfig      |    2 ++
 kernel/bc/Kconfig |   25 +++++++++++++++++++++++++
 2 files changed, 27 insertions(+)

--- ./init/Kconfig.bckm 2006-07-10 12:39:10.000000000 +0400
+++ ./init/Kconfig 2006-07-28 14:10:41.000000000 +0400
@@ -222,6 +222,8 @@ source "crypto/Kconfig"
```

```
   Say N if unsure.

+source "kernel/bc/Kconfig"
+
 config SYSCTL
  bool

--- ./kernel/bc/Kconfig.bckm 2006-07-28 13:07:38.000000000 +0400
+++ ./kernel/bc/Kconfig 2006-07-28 13:09:51.000000000 +0400
@@ -0,0 +1,25 @@
+#
+# Resource beancounters (BC)
+#
+# Copyright (C) 2006 OpenVZ. SWsoft Inc
+
+menu "User resources"
+
+config BEANCOUNTERS
+ bool "Enable resource accounting/control"
+ default n
+ help
+         When Y this option provides accounting and allows to configure
+         limits for user's consumption of exhaustible system resources.
+         The most important resource controlled by this patch is unswappable
+         memory (either mlock'ed or used by internal kernel structures and
+         buffers). The main goal of this patch is to protect processes
+         from running short of important resources because of an accidental
+         misbehavior of processes or malicious activity aiming to ``kill''
+         the system. It's worth to mention that resource limits configured
+         by setrlimit(2) do not give an acceptable level of protection
+         because they cover only small fraction of resources and work on a
+         per-process basis.  Per-process accounting doesn't prevent malicious
+         users from spawning a lot of resource-consuming processes.
+
+endmenu
```

Subject: [PATCH 3/7] BC: beancounters core (API)
Posted by dev on Tue, 29 Aug 2006 14:50:18 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 | 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; /* minumum 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];
+};
+
+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 */
+
+#define BC_HASH_BITS  8
+#define BC_HASH_SIZE  (1 << BC_HASH_BITS)
+
+#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;
+}
+
+int __must_check bc_charge_locked(struct beancounter *bc,
+  int res, unsigned long val, enum bc_severity strict);
+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);
+
+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[];
+
+#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 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 2006-08-28 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-$(CONFIG_STACKTRACE) += stacktrace.o
 obj-y += time/
+obj-y += bc/
 obj-$(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;
+
+ 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);
+
+/*
+ * 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;
+
+ 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);
+  return 0;
+ default:
+  BUG();
+ }
+
+ bc->bc_parms[resource].failcnt++;
+ bc->bc_parms[resource].held -= val;
+ return -ENOMEM;
+}
+EXPORT_SYMBOL(bc_charge_locked);
+
+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);
+
+/* 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);
+}
+EXPORT_SYMBOL(bc_uncharge_locked);
+
+void bc_uncharge(struct beancounter *bc, int resource, unsigned long val)
+{
```

```
+ unsigned long flags;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ bc_uncharge_locked(bc, resource, val);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+}
+EXPORT_SYMBOL(bc_uncharge);
+
+/*
+ * 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);
+}
```

Subject: [PATCH 4/7] BC: context inheriting and changing
Posted by dev on Tue, 29 Aug 2006 14:51:35 GMT
View Forum Message <> Reply to Message

Contains code responsible for setting BC on task,
it's inheriting and setting host context in interrupts.

Task references 2 beancounters:
1. exec_bc: current context. all resources are
        charged to this beancounter.
3. fork_bc: beancounter which is inherited by
        task's children on fork

Signed-off-by: Pavel Emelianov <xemul@sw.ru>
Signed-off-by: Kirill Korotaev <dev@sw.ru>

```
 include/bc/task.h       |   59 ++++++
```

```
 kernel/bc/beancounter.c |    3 ++
 kernel/bc/misc.c        |   32 ++++++++++++++++++++++++++++
 kernel/fork.c           |   17 ++++++++++++-
 kernel/irq/handle.c     |    9 +++++++
 kernel/softirq.c        |    8 ++++++
 8 files changed, 132 insertions(+), 2 deletions(-)

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/task.h 2006-08-28 12:54:48.000000000 +0400
@@ -0,0 +1,59 @@
+/*
+ *  include/bc/task.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#ifndef __BC_TASK_H_
+#define __BC_TASK_H_
+
+struct beancounter;
+
+struct task_beancounter {
+ struct beancounter *exec_bc;
+ struct beancounter *fork_bc;
+};
+
+#ifdef CONFIG_BEANCOUNTERS
+
+#define get_exec_bc()  (current->task_bc.exec_bc)
+#define set_exec_bc(new)   \
+ ({     \
+   struct beancounter *old; \
+   struct task_beancounter *tbc; \
+   tbc = &current->task_bc; \
+   old = tbc->exec_bc;  \
+   tbc->exec_bc = new;  \
+   old;    \
+  })
+#define reset_exec_bc(old, exp)   \
+ do {     \
+  struct task_beancounter *tbc; \
+  tbc = &current->task_bc; \
+  BUG_ON(tbc->exec_bc != exp); \
+  tbc->exec_bc = old;  \
+ } while (0)
+
+int __must_check bc_task_charge(struct task_struct *parent,
```

```
+  struct task_struct *new);
+void bc_task_uncharge(struct task_struct *tsk);
+
+#else
+
+#define get_exec_bc()  (NULL)
+#define set_exec_bc(new) (NULL)
+#define reset_exec_bc(new, exp) do { } while (0)
+
+static inline __must_check int bc_task_charge(struct task_struct *parent,
+  struct task_struct *new)
+{
+ return 0;
+}
+
+static inline void bc_task_uncharge(struct task_struct *tsk)
+{
+}
+
+#endif
+#endif
--- ./include/linux/sched.h.bctask 2006-08-28 12:20:13.000000000 +0400
+++ ./include/linux/sched.h 2006-08-28 12:52:11.000000000 +0400
@@ -83,6 +83,8 @@ struct sched_param {
 #include <linux/timer.h>
 #include <linux/hrtimer.h>

+#include <bc/task.h>
+
 #include <asm/processor.h>

 struct exec_domain;
@@ -1048,6 +1050,9 @@ struct task_struct {
  spinlock_t delays_lock;
  struct task_delay_info *delays;
 #endif
+#ifdef CONFIG_BEANCOUNTERS
+ struct task_beancounter task_bc;
+#endif
 };

 static inline pid_t process_group(struct task_struct *tsk)
--- ./kernel/bc/Makefile.bctask 2006-08-28 12:43:34.000000000 +0400
+++ ./kernel/bc/Makefile 2006-08-28 12:52:11.000000000 +0400
@@ -5,3 +5,4 @@
 #

 obj-$(CONFIG_BEANCOUNTERS) += beancounter.o
```

```
+obj-$(CONFIG_BEANCOUNTERS) += misc.o
--- ./kernel/bc/beancounter.c.bctask 2006-08-28 12:49:07.000000000 +0400
+++ ./kernel/bc/beancounter.c 2006-08-28 12:52:11.000000000 +0400
@@ -238,6 +238,9 @@ void __init bc_init_early(void)
  spin_lock_init(&bc_hash_lock);
  slot = &bc_hash[bc_hash_fn(bc->bc_id)];
  hlist_add_head(&bc->hash, slot);
+
+ current->task_bc.exec_bc = get_beancounter(bc);
+ current->task_bc.fork_bc = get_beancounter(bc);
 }

 void __init bc_init_late(void)
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/misc.c 2006-08-28 12:52:11.000000000 +0400
@@ -0,0 +1,32 @@
+/*
+ * kernel/bc/misc.c
+ *
+ * Copyright (C) 2006 OpenVZ. SWsoft Inc.
+ *
+ */
+
+#include <linux/sched.h>
+
+#include <bc/beancounter.h>
+#include <bc/task.h>
+
+int bc_task_charge(struct task_struct *parent, struct task_struct *new)
+{
+ struct task_beancounter *old_bc;
+ struct task_beancounter *new_bc;
+ struct beancounter *bc;
+
+ old_bc = &parent->task_bc;
+ new_bc = &new->task_bc;
+
+ bc = old_bc->fork_bc;
+ new_bc->exec_bc = get_beancounter(bc);
+ new_bc->fork_bc = get_beancounter(bc);
+ return 0;
+}
+
+void bc_task_uncharge(struct task_struct *tsk)
+{
+ put_beancounter(tsk->task_bc.exec_bc);
+ put_beancounter(tsk->task_bc.fork_bc);
+}
```

```
--- ./kernel/fork.c.bctask 2006-08-28 12:20:13.000000000 +0400
+++ ./kernel/fork.c 2006-08-28 12:52:11.000000000 +0400
@@ -49,6 +49,8 @@
 #include <linux/taskstats_kern.h>
 #include <linux/random.h>

+#include <bc/task.h>
+
 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
 #include <asm/uaccess.h>
@@ -103,12 +105,18 @@ kmem_cache_t *vm_area_cachep;
 /* SLAB cache for mm_struct structures (tsk->mm) */
 static kmem_cache_t *mm_cachep;

-void free_task(struct task_struct *tsk)
+static void __free_task(struct task_struct *tsk)
 {
  free_thread_info(tsk->thread_info);
  rt_mutex_debug_task_free(tsk);
  free_task_struct(tsk);
 }
+
+void free_task(struct task_struct *tsk)
+{
+ bc_task_uncharge(tsk);
+ __free_task(tsk);
+}
 EXPORT_SYMBOL(free_task);

 void __put_task_struct(struct task_struct *tsk)
@@ -983,6 +991,9 @@ static struct task_struct *copy_process(
  if (!p)
   goto fork_out;

+ if (bc_task_charge(current, p))
+  goto bad_charge;
+
 #ifdef CONFIG_TRACE_IRQFLAGS
  DEBUG_LOCKS_WARN_ON(!p->hardirqs_enabled);
  DEBUG_LOCKS_WARN_ON(!p->softirqs_enabled);
@@ -1293,7 +1304,9 @@ bad_fork_cleanup_count:
  atomic_dec(&p->user->processes);
  free_uid(p->user);
 bad_fork_free:
- free_task(p);
+ bc_task_uncharge(p);
+bad_charge:
```

```
+ __free_task(p);
 fork_out:
  return ERR_PTR(retval);
 }
--- ./kernel/irq/handle.c.bctask 2006-07-10 12:39:20.000000000 +0400
+++ ./kernel/irq/handle.c 2006-08-28 12:52:11.000000000 +0400
@@ -16,6 +16,9 @@
 #include <linux/interrupt.h>
 #include <linux/kernel_stat.h>

+#include <bc/beancounter.h>
+#include <bc/task.h>
+
 #include "internals.h"

 /**
@@ -166,6 +169,9 @@ fastcall unsigned int __do_IRQ(unsigned
  struct irq_desc *desc = irq_desc + irq;
  struct irqaction *action;
  unsigned int status;
+ struct beancounter *bc;
+
+ bc = set_exec_bc(&init_bc);

  kstat_this_cpu.irqs[irq]++;
  if (CHECK_IRQ_PER_CPU(desc->status)) {
@@ -178,6 +184,8 @@ fastcall unsigned int __do_IRQ(unsigned
    desc->chip->ack(irq);
   action_ret = handle_IRQ_event(irq, regs, desc->action);
   desc->chip->end(irq);
+
+  reset_exec_bc(bc, &init_bc);
   return 1;
  }

@@ -246,6 +254,7 @@ out:
  desc->chip->end(irq);
  spin_unlock(&desc->lock);

+ reset_exec_bc(bc, &init_bc);
  return 1;
 }

--- ./kernel/softirq.c.bctask 2006-08-28 12:20:13.000000000 +0400
+++ ./kernel/softirq.c 2006-08-28 12:52:11.000000000 +0400
@@ -18,6 +18,9 @@
 #include <linux/rcupdate.h>
 #include <linux/smp.h>
```

```
+#include <bc/beancounter.h>
+#include <bc/task.h>
+
 #include <asm/irq.h>
 /*
    - No shared variables, all the data are CPU local.
@@ -209,6 +212,9 @@ asmlinkage void __do_softirq(void)
  __u32 pending;
  int max_restart = MAX_SOFTIRQ_RESTART;
  int cpu;
+ struct beancounter *bc;
+
+ bc = set_exec_bc(&init_bc);

  pending = local_softirq_pending();
  account_system_vtime(current);
@@ -247,6 +253,8 @@ restart:

  account_system_vtime(current);
  _local_bh_enable();
+
+ reset_exec_bc(bc, &init_bc);
 }

 #ifndef __ARCH_HAS_DO_SOFTIRQ
```

Subject: [PATCH 5/7] BC: user interface (syscalls)
Posted by dev on Tue, 29 Aug 2006 14:52:39 GMT
View Forum Message <> Reply to Message

Add the following system calls for BC management:
1. sys_get_bcid - get current BC id
2. sys_set_bcid - change exec_ and fork_ BCs on current
3. sys_set_bclimit - set limits for resources consumtions
4. sys_get_bcstat - return br_resource_parm on resource

Signed-off-by: Pavel Emelianov <xemul@sw.ru>
Signed-off-by: Kirill Korotaev <dev@sw.ru>

```
 arch/i386/kernel/syscall_table.S |    4 +
 arch/ia64/kernel/entry.S         |    4 +
 arch/sparc/kernel/entry.S        |    2
 arch/sparc/kernel/systbls.S      |    6 +
 arch/sparc64/kernel/entry.S      |    2
```

```
 arch/sparc64/kernel/systbls.S    |   10 ++-
 include/asm-i386/unistd.h        |    6 +
 include/asm-ia64/unistd.h        |    6 +
 include/asm-powerpc/systbl.h     |    4 +
 include/asm-powerpc/unistd.h     |    6 +
 include/asm-sparc/unistd.h       |    4 +
 include/asm-sparc64/unistd.h     |    4 +
 include/asm-x86_64/unistd.h      |   10 ++-
 kernel/bc/Makefile               |    1
 kernel/bc/sys.c                  |  120 ++++++++++++++++++++++++++++++++++++++++
 kernel/sys_ni.c                  |    6 +
 16 files changed, 186 insertions(+), 9 deletions(-)

--- ./arch/i386/kernel/syscall_table.S.ve3 2006-08-21 13:15:37.000000000 +0400
+++ ./arch/i386/kernel/syscall_table.S 2006-08-21 14:15:47.000000000 +0400
@@ -318,3 +318,7 @@ ENTRY(sys_call_table)
  .long sys_vmsplice
  .long sys_move_pages
  .long sys_getcpu
+ .long sys_get_bcid
+ .long sys_set_bcid  /* 320 */
+ .long sys_set_bclimit
+ .long sys_get_bcstat
--- ./arch/ia64/kernel/entry.S.ve3 2006-08-21 13:15:37.000000000 +0400
+++ ./arch/ia64/kernel/entry.S 2006-08-21 14:17:07.000000000 +0400
@@ -1610,5 +1610,9 @@ sys_call_table:
  data8 sys_sync_file_range  // 1300
  data8 sys_tee
  data8 sys_vmsplice
+ data8 sys_get_bcid
+ data8 sys_set_bcid
+ data8 sys_set_bclimit   // 1305
+ data8 sys_get_bcstat

  .org sys_call_table + 8*NR_syscalls // guard against failures to increase NR_syscalls
--- ./arch/sparc/kernel/entry.S.ve3 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/sparc/kernel/entry.S 2006-08-21 14:29:44.000000000 +0400
@@ -37,7 +37,7 @@

 #define curptr      g6

-#define NR_SYSCALLS 300      /* Each OS is different... */
+#define NR_SYSCALLS 304      /* Each OS is different... */

 /* These are just handy. */
 #define _SV save %sp, -STACKFRAME_SZ, %sp
--- ./arch/sparc/kernel/systbls.S.ve3 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/sparc/kernel/systbls.S 2006-08-21 14:30:43.000000000 +0400
```

```
@@ -78,7 +78,8 @@ sys_call_table:
 /*285*/ .long sys_mkdirat, sys_mknodat, sys_fchownat, sys_futimesat, sys_fstatat64
 /*290*/ .long sys_unlinkat, sys_renameat, sys_linkat, sys_symlinkat, sys_readlinkat
 /*295*/ .long sys_fchmodat, sys_faccessat, sys_pselect6, sys_ppoll, sys_unshare
-/*300*/ .long sys_set_robust_list, sys_get_robust_list
+/*300*/ .long sys_set_robust_list, sys_get_robust_list, sys_get_bcid, sys_set_bcid,
sys_set_bclimit
+/*305*/ .long sys_get_bcstat

 #ifdef CONFIG_SUNOS_EMUL
  /* Now the SunOS syscall table. */
@@ -192,4 +193,7 @@ sunos_sys_table:
  .long sunos_nosys, sunos_nosys, sunos_nosys
  .long sunos_nosys, sunos_nosys, sunos_nosys

+ .long sunos_nosys, sunos_nosys, sunos_nosys,
+ .long sunos_nosys
+
 #endif
--- ./arch/sparc64/kernel/entry.S.ve3 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/sparc64/kernel/entry.S 2006-08-21 14:29:56.000000000 +0400
@@ -25,7 +25,7 @@

 #define curptr      g6

-#define NR_SYSCALLS 300      /* Each OS is different... */
+#define NR_SYSCALLS 304      /* Each OS is different... */

  .text
  .align  32
--- ./arch/sparc64/kernel/systbls.S.ve3 2006-07-10 12:39:11.000000000 +0400
+++ ./arch/sparc64/kernel/systbls.S 2006-08-21 14:32:26.000000000 +0400
@@ -79,7 +79,8 @@ sys_call_table32:
  .word sys_mkdirat, sys_mknodat, sys_fchownat, compat_sys_futimesat, compat_sys_fstatat64
 /*290*/ .word sys_unlinkat, sys_renameat, sys_linkat, sys_symlinkat, sys_readlinkat
  .word sys_fchmodat, sys_faccessat, compat_sys_pselect6, compat_sys_ppoll, sys_unshare
-/*300*/ .word compat_sys_set_robust_list, compat_sys_get_robust_list
+/*300*/ .word compat_sys_set_robust_list, compat_sys_get_robust_list, sys_nis_syscall,
sys_nis_syscall, sys_nis_syscall
+ .word sys_nis_syscall

 #endif /* CONFIG_COMPAT */

@@ -149,7 +150,9 @@ sys_call_table:
  .word sys_mkdirat, sys_mknodat, sys_fchownat, sys_futimesat, sys_fstatat64
 /*290*/ .word sys_unlinkat, sys_renameat, sys_linkat, sys_symlinkat, sys_readlinkat
  .word sys_fchmodat, sys_faccessat, sys_pselect6, sys_ppoll, sys_unshare
-/*300*/ .word sys_set_robust_list, sys_get_robust_list
+/*300*/ .word sys_set_robust_list, sys_get_robust_list
```

```
+/*300*/ .word sys_set_robust_list, sys_get_robust_list, sys_get_bcid, sys_set_bcid,
sys_set_bclimit
+ .word sys_get_bcstat
+

 #if defined(CONFIG_SUNOS_EMUL) || defined(CONFIG_SOLARIS_EMUL) || \
     defined(CONFIG_SOLARIS_EMUL_MODULE)
@@ -263,4 +266,7 @@ sunos_sys_table:
  .word sunos_nosys, sunos_nosys, sunos_nosys
  .word sunos_nosys, sunos_nosys, sunos_nosys
  .word sunos_nosys, sunos_nosys, sunos_nosys
+
+ .word sunos_nosys, sunos_nosys, sunos_nosys
+ .word sunos_nosys
 #endif
--- ./include/asm-i386/unistd.h.ve3 2006-08-21 13:15:39.000000000 +0400
+++ ./include/asm-i386/unistd.h 2006-08-21 14:22:53.000000000 +0400
@@ -324,10 +324,14 @@
 #define __NR_vmsplice  316
 #define __NR_move_pages  317
 #define __NR_getcpu  318
+#define __NR_get_bcid  319
+#define __NR_set_bcid  320
+#define __NR_set_bclimit 321
+#define __NR_get_bcstat  322

 #ifdef __KERNEL__

-#define NR_syscalls 318
+#define NR_syscalls 323
 #include <linux/err.h>

 /*
--- ./include/asm-ia64/unistd.h.ve3 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-ia64/unistd.h 2006-08-21 14:24:29.000000000 +0400
@@ -291,11 +291,15 @@
 #define __NR_sync_file_range  1300
 #define __NR_tee   1301
 #define __NR_vmsplice   1302
+#define __NR_get_bcid   1303
+#define __NR_set_bcid   1304
+#define __NR_set_bclimit  1305
+#define __NR_get_bcstat   1306

 #ifdef __KERNEL__


-#define NR_syscalls   279 /* length of syscall table */
```

```
+#define NR_syscalls   283 /* length of syscall table */

 #define __ARCH_WANT_SYS_RT_SIGACTION

--- ./include/asm-powerpc/systbl.h.ve3 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-powerpc/systbl.h 2006-08-21 14:28:46.000000000 +0400
@@ -304,3 +304,7 @@ SYSCALL_SPU(fchmodat)
 SYSCALL_SPU(faccessat)
 COMPAT_SYS_SPU(get_robust_list)
 COMPAT_SYS_SPU(set_robust_list)
+SYSCALL(sys_get_bcid)
+SYSCALL(sys_set_bcid)
+SYSCALL(sys_set_bclimit)
+SYSCALL(sys_get_bcstat)
--- ./include/asm-powerpc/unistd.h.ve3 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-powerpc/unistd.h 2006-08-21 14:28:24.000000000 +0400
@@ -323,10 +323,14 @@
 #define __NR_faccessat  298
 #define __NR_get_robust_list 299
 #define __NR_set_robust_list 300
+#define __NR_get_bcid  301
+#define __NR_set_bcid  302
+#define __NR_set_bclimit 303
+#define __NR_get_bcstat  304

 #ifdef __KERNEL__

-#define __NR_syscalls  301
+#define __NR_syscalls  305

 #define __NR__exit __NR_exit
 #define NR_syscalls __NR_syscalls
--- ./include/asm-sparc/unistd.h.ve3 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-sparc/unistd.h 2006-08-21 14:33:20.000000000 +0400
@@ -318,6 +318,10 @@
 #define __NR_unshare  299
 #define __NR_set_robust_list 300
 #define __NR_get_robust_list 301
+#define __NR_get_bcid  302
+#define __NR_set_bcid  303
+#define __NR_set_bclimit 304
+#define __NR_get_bcstat  305

 #ifdef __KERNEL__
 /* WARNING: You MAY NOT add syscall numbers larger than 301, since
--- ./include/asm-sparc64/unistd.h.ve3 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-sparc64/unistd.h 2006-08-21 14:34:10.000000000 +0400
@@ -320,6 +320,10 @@
```

```
 #define __NR_unshare  299
 #define __NR_set_robust_list 300
 #define __NR_get_robust_list 301
+#define __NR_get_bcid  302
+#define __NR_set_bcid  303
+#define __NR_set_bclimit 304
+#define __NR_get_bcstat  305

 #ifdef __KERNEL__
 /* WARNING: You MAY NOT add syscall numbers larger than 301, since
--- ./include/asm-x86_64/unistd.h.ve3 2006-08-21 13:15:39.000000000 +0400
+++ ./include/asm-x86_64/unistd.h 2006-08-21 14:35:19.000000000 +0400
@@ -619,10 +619,18 @@ __SYSCALL(__NR_sync_file_range, sys_sync
 __SYSCALL(__NR_vmsplice, sys_vmsplice)
 #define __NR_move_pages  279
 __SYSCALL(__NR_move_pages, sys_move_pages)
+#define __NR_get_bcid  280
+__SYSCALL(__NR_get_bcid, sys_get_bcid)
+#define __NR_set_bcid  281
+__SYSCALL(__NR_set_bcid, sys_set_bcid)
+#define __NR_set_bclimit 282
+__SYSCALL(__NR_set_bclimit, sys_set_bclimit)
+#define __NR_get_bcstat  283
+__SYSCALL(__NR_get_bcstat, sys_get_bcstat)

 #ifdef __KERNEL__

-#define __NR_syscall_max __NR_move_pages
+#define __NR_syscall_max __NR_get_bcstat
 #include <linux/err.h>

 #ifndef __NO_STUBS
--- ./kernel/sys_ni.c.ve3 2006-07-10 12:39:20.000000000 +0400
+++ ./kernel/sys_ni.c 2006-08-21 14:12:49.000000000 +0400
@@ -134,3 +134,9 @@ cond_syscall(sys_madvise);
 cond_syscall(sys_mremap);
 cond_syscall(sys_remap_file_pages);
 cond_syscall(compat_sys_move_pages);
+
+/* user resources syscalls */
+cond_syscall(sys_set_bcid);
+cond_syscall(sys_get_bcid);
+cond_syscall(sys_set_bclimit);
+cond_syscall(sys_get_bcstat);
--- ./kernel/bc/Makefile.ve3 2006-08-21 13:49:49.000000000 +0400
+++ ./kernel/bc/Makefile 2006-08-21 13:55:39.000000000 +0400
@@ -6,3 +6,4 @@
```

```
 obj-$(CONFIG_BEANCOUNTERS) += beancounter.o
 obj-$(CONFIG_BEANCOUNTERS) += misc.o
+obj-$(CONFIG_BEANCOUNTERS) += sys.o
--- ./kernel/bc/sys.c.ve3 2006-08-21 13:49:49.000000000 +0400
+++ ./kernel/bc/sys.c 2006-08-21 14:43:04.000000000 +0400
@@ -0,0 +1,120 @@
+/*
+ *  kernel/bc/sys.c
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#include <linux/sched.h>
+#include <asm/uaccess.h>
+
+#include <bc/beancounter.h>
+#include <bc/task.h>
+
+asmlinkage long sys_get_bcid(void)
+{
+ struct beancounter *bc;
+
+ bc = get_exec_bc();
+ return bc->bc_id;
+}
+
+asmlinkage long sys_set_bcid(bcid_t id)
+{
+ int error;
+ struct beancounter *bc;
+ struct task_beancounter *task_bc;
+
+ task_bc = &current->task_bc;
+
+ /* You may only set an bc as root */
+ error = -EPERM;
+ if (!capable(CAP_SETUID))
+  goto out;
+
+ /* Ok - set up a beancounter entry for this user */
+ error = -ENOMEM;
+ bc = beancounter_findcreate(id, BC_ALLOC);
+ if (bc == NULL)
+  goto out;
+
+ /* install bc */
+ put_beancounter(task_bc->exec_bc);
```

```
+ task_bc->exec_bc = bc;
+ put_beancounter(task_bc->fork_bc);
+ task_bc->fork_bc = get_beancounter(bc);
+ error = 0;
+out:
+ return error;
+}
+
+asmlinkage long sys_set_bclimit(bcid_t id, unsigned long resource,
+  unsigned long __user *limits)
+{
+ int error;
+ unsigned long flags;
+ struct beancounter *bc;
+ unsigned long new_limits[2];
+
+ error = -EPERM;
+ if(!capable(CAP_SYS_RESOURCE))
+  goto out;
+
+ error = -EINVAL;
+ if (resource >= BC_RESOURCES)
+  goto out;
+
+ error = -EFAULT;
+ if (copy_from_user(&new_limits, limits, sizeof(new_limits)))
+  goto out;
+
+ error = -EINVAL;
+ if (new_limits[0] > BC_MAXVALUE || new_limits[1] > BC_MAXVALUE ||
+    new_limits[0] > new_limits[1])
+  goto out;
+
+ error = -ENOENT;
+ bc = beancounter_findcreate(id, BC_LOOKUP);
+ if (bc == NULL)
+  goto out;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ bc->bc_parms[resource].barrier = new_limits[0];
+ bc->bc_parms[resource].limit = new_limits[1];
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+
+ put_beancounter(bc);
+ error = 0;
+out:
+ return error;
+}
```

```
+
+int sys_get_bcstat(bcid_t id, unsigned long resource,
+  struct bc_resource_parm __user *uparm)
+{
+ int error;
+ unsigned long flags;
+ struct beancounter *bc;
+ struct bc_resource_parm parm;
+
+ error = -EINVAL;
+ if (resource >= BC_RESOURCES)
+  goto out;
+
+ error = -ENOENT;
+ bc = beancounter_findcreate(id, BC_LOOKUP);
+ if (bc == NULL)
+  goto out;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ parm = bc->bc_parms[resource];
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+ put_beancounter(bc);
+
+ error = 0;
+ if (copy_to_user(uparm, &parm, sizeof(parm)))
+  error = -EFAULT;
+
+out:
+ return error;
+}
```

Subject: [PATCH 6/7] BC: kernel memory (core)
Posted by dev on Tue, 29 Aug 2006 14:55:19 GMT
View Forum Message <> Reply to Message

Introduce BC_KMEMSIZE resource which accounts kernel
objects allocated by task's request.

Reference to BC is kept on struct page or slab object.
For slabs each struct slab contains a set of pointers
corresponding objects are charged to.

Allocation charge rules:
1. Pages - if allocation is performed with __GFP_BC flag - page
   is charged to current's exec_bc.
2. Slabs - kmem_cache may be created with SLAB_BC flag - in this
   case each allocation is charged. Caches used by kmalloc are

created with SLAB_BC | SLAB_BC_NOCHARGE flags. In this case
only __GFP_BC allocations are charged.

Signed-off-by: Pavel Emelianov <xemul@sw.ru>
Signed-off-by: Kirill Korotaev <dev@sw.ru>

```
 include/bc/beancounter.h |    4 +
 include/bc/kmem.h        |   46 +++++++++++++++++
 include/linux/gfp.h      |    8 ++-
 include/linux/mm.h       |    6 ++
 include/linux/slab.h     |    4 +
 include/linux/vmalloc.h  |    1
 kernel/bc/Makefile       |    1
 kernel/bc/beancounter.c  |    3 +
 kernel/bc/kmem.c         |   85 ++++++++++++++++++++++++++++++++++
 mm/mempool.c             |    2
 mm/page_alloc.c          |   11 ++++
 mm/slab.c                |  121 +++++++++++++++++++++++++++++++++++++++---------
 mm/vmalloc.c             |    6 ++
 13 files changed, 273 insertions(+), 25 deletions(-)

--- ./include/bc/beancounter.h.bckmem 2006-08-28 12:47:52.000000000 +0400
+++ ./include/bc/beancounter.h 2006-08-28 12:59:28.000000000 +0400
@@ -12,7 +12,9 @@
  * Resource list.
  */

-#define BC_RESOURCES 0
+#define BC_KMEMSIZE 0
+
+#define BC_RESOURCES 1

 struct bc_resource_parm {
  unsigned long barrier; /* A barrier over which resource allocations
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/kmem.h 2006-08-28 13:00:43.000000000 +0400
@@ -0,0 +1,46 @@
+/*
+ *  include/bc/kmem.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#ifndef __BC_KMEM_H_
+#define __BC_KMEM_H_
```

```
+
+/*
+ * BC_KMEMSIZE accounting
+ */
+
+struct mm_struct;
+struct page;
+struct beancounter;
+
+#ifdef CONFIG_BEANCOUNTERS
+int __must_check bc_page_charge(struct page *page, int order, gfp_t flags);
+void bc_page_uncharge(struct page *page, int order);
+
+int __must_check bc_slab_charge(kmem_cache_t *cachep, void *obj, gfp_t flags);
+void bc_slab_uncharge(kmem_cache_t *cachep, void *obj);
+#else
+static inline int __must_check bc_page_charge(struct page *page,
+  int order, gfp_t flags)
+{
+ return 0;
+}
+
+static inline void bc_page_uncharge(struct page *page, int order)
+{
+}
+
+static inline int __must_check bc_slab_charge(kmem_cache_t *cachep,
+  void *obj, gfp_t flags)
+{
+ return 0;
+}
+
+static inline void bc_slab_uncharge(kmem_cache_t *cachep, void *obj)
+{
+}
+#endif
+#endif /* __BC_SLAB_H_ */
--- ./include/linux/gfp.h.bckmem 2006-08-28 12:20:13.000000000 +0400
+++ ./include/linux/gfp.h 2006-08-28 12:59:28.000000000 +0400
@@ -46,15 +46,18 @@ struct vm_area_struct;
 #define __GFP_NOMEMALLOC ((__force gfp_t)0x10000u) /* Don't use emergency reserves */
 #define __GFP_HARDWALL   ((__force gfp_t)0x20000u) /* Enforce hardwall cpuset memory
allocs */
 #define __GFP_THISNODE ((__force gfp_t)0x40000u)/* No fallback, no policies */
+#define __GFP_BC  ((__force gfp_t)0x80000u) /* Charge allocation with BC */
+#define __GFP_BC_LIMIT ((__force gfp_t)0x100000u) /* Charge against BC limit */

-#define __GFP_BITS_SHIFT 20 /* Room for 20 __GFP_FOO bits */
```

```
+#define __GFP_BITS_SHIFT 21 /* Room for 21 __GFP_FOO bits */
 #define __GFP_BITS_MASK ((__force gfp_t)((1 << __GFP_BITS_SHIFT) - 1))

 /* if you forget to add the bitmask here kernel will crash, period */
 #define GFP_LEVEL_MASK (__GFP_WAIT|__GFP_HIGH|__GFP_IO|__GFP_FS| \
    __GFP_COLD|__GFP_NOWARN|__GFP_REPEAT| \
    __GFP_NOFAIL|__GFP_NORETRY|__GFP_NO_GROW|__GFP_COMP| \
-   __GFP_NOMEMALLOC|__GFP_HARDWALL|__GFP_THISNODE)
+   __GFP_NOMEMALLOC|__GFP_HARDWALL|__GFP_THISNODE| \
+   __GFP_BC|__GFP_BC_LIMIT)

 /* This equals 0, but use constants in case they ever change */
 #define GFP_NOWAIT (GFP_ATOMIC & ~__GFP_HIGH)
@@ -63,6 +66,7 @@ struct vm_area_struct;
 #define GFP_NOIO (__GFP_WAIT)
 #define GFP_NOFS (__GFP_WAIT | __GFP_IO)
 #define GFP_KERNEL (__GFP_WAIT | __GFP_IO | __GFP_FS)
+#define GFP_KERNEL_BC (__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_BC)
 #define GFP_USER (__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_HARDWALL)
 #define GFP_HIGHUSER (__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_HARDWALL | \
     __GFP_HIGHMEM)
--- ./include/linux/mm.h.bckmem 2006-08-28 12:20:13.000000000 +0400
+++ ./include/linux/mm.h 2006-08-28 12:59:28.000000000 +0400
@@ -274,8 +274,14 @@ struct page {
  unsigned int gfp_mask;
  unsigned long trace[8];
 #endif
+#ifdef CONFIG_BEANCOUNTERS
+ union {
+  struct beancounter *page_bc;
+ } bc;
+#endif
 };

+#define page_bc(page)   ((page)->bc.page_bc)
 #define page_private(page)  ((page)->private)
 #define set_page_private(page, v) ((page)->private = (v))

--- ./include/linux/slab.h.bckmem 2006-08-28 12:20:13.000000000 +0400
+++ ./include/linux/slab.h 2006-08-28 12:59:28.000000000 +0400
@@ -46,6 +46,8 @@ typedef struct kmem_cache kmem_cache_t;
 #define SLAB_PANIC  0x00040000UL /* panic if kmem_cache_create() fails */
 #define SLAB_DESTROY_BY_RCU 0x00080000UL /* defer freeing pages to RCU */
 #define SLAB_MEM_SPREAD  0x00100000UL /* Spread some memory over cpuset */
+#define SLAB_BC  0x00200000UL /* Account with BC */
+#define SLAB_BC_NOCHARGE 0x00400000UL /* Explicit accounting */

 /* flags passed to a constructor func */
```

```
 #define SLAB_CTOR_CONSTRUCTOR 0x001UL  /* if not set, then deconstructor */
@@ -291,6 +293,8 @@ extern kmem_cache_t *fs_cachep;
 extern kmem_cache_t *sighand_cachep;
 extern kmem_cache_t *bio_cachep;

+struct beancounter;
+struct beancounter **kmem_cache_bcp(kmem_cache_t *cachep, void *obj);
 #endif /* __KERNEL__ */

 #endif /* _LINUX_SLAB_H */
--- ./include/linux/vmalloc.h.bckmem 2006-08-28 12:20:13.000000000 +0400
+++ ./include/linux/vmalloc.h 2006-08-28 12:59:28.000000000 +0400
@@ -36,6 +36,7 @@ struct vm_struct {
  * Highlevel APIs for driver use
  */
 extern void *vmalloc(unsigned long size);
+extern void *vmalloc_bc(unsigned long size);
 extern void *vmalloc_user(unsigned long size);
 extern void *vmalloc_node(unsigned long size, int node);
 extern void *vmalloc_exec(unsigned long size);
--- ./kernel/bc/Makefile.bckmem 2006-08-28 12:58:27.000000000 +0400
+++ ./kernel/bc/Makefile 2006-08-28 12:59:28.000000000 +0400
@@ -7,3 +7,4 @@
 obj-$(CONFIG_BEANCOUNTERS) += beancounter.o
 obj-$(CONFIG_BEANCOUNTERS) += misc.o
 obj-$(CONFIG_BEANCOUNTERS) += sys.o
+obj-$(CONFIG_BEANCOUNTERS) += kmem.o
--- ./kernel/bc/beancounter.c.bckmem 2006-08-28 12:52:11.000000000 +0400
+++ ./kernel/bc/beancounter.c 2006-08-28 12:59:28.000000000 +0400
@@ -20,6 +20,7 @@ static void init_beancounter_struct(stru
 struct beancounter init_bc;

 const char *bc_rnames[] = {
+ "kmemsize", /* 0 */
 };

 static struct hlist_head bc_hash[BC_HASH_SIZE];
@@ -221,6 +222,8 @@ static void init_beancounter_syslimits(s
 {
  int k;

+ bc->bc_parms[BC_KMEMSIZE].limit = 32 * 1024 * 1024;
+
  for (k = 0; k < BC_RESOURCES; k++)
   bc->bc_parms[k].barrier = bc->bc_parms[k].limit;
 }
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/kmem.c 2006-08-28 12:59:28.000000000 +0400
```

```
@@ -0,0 +1,85 @@
+/*
+ *  kernel/bc/kmem.c
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#include <linux/sched.h>
+#include <linux/gfp.h>
+#include <linux/slab.h>
+#include <linux/mm.h>
+
+#include <bc/beancounter.h>
+#include <bc/kmem.h>
+#include <bc/task.h>
+
+/*
+ * Slab accounting
+ */
+
+int bc_slab_charge(kmem_cache_t *cachep, void *objp, gfp_t flags)
+{
+ unsigned int size;
+ struct beancounter *bc, **slab_bcp;
+
+ bc = get_exec_bc();
+
+ size = kmem_cache_size(cachep);
+ if (bc_charge(bc, BC_KMEMSIZE, size,
+   (flags & __GFP_BC_LIMIT ? BC_LIMIT : BC_BARRIER)))
+  return -ENOMEM;
+
+ slab_bcp = kmem_cache_bcp(cachep, objp);
+ *slab_bcp = get_beancounter(bc);
+ return 0;
+}
+
+void bc_slab_uncharge(kmem_cache_t *cachep, void *objp)
+{
+ unsigned int size;
+ struct beancounter *bc, **slab_bcp;
+
+ slab_bcp = kmem_cache_bcp(cachep, objp);
+ if (*slab_bcp == NULL)
+  return;
+
+ bc = *slab_bcp;
```

```
+ size = kmem_cache_size(cachep);
+ bc_uncharge(bc, BC_KMEMSIZE, size);
+ put_beancounter(bc);
+ *slab_bcp = NULL;
+}
+
+/*
+ * Pages accounting
+ */
+
+int bc_page_charge(struct page *page, int order, gfp_t flags)
+{
+ struct beancounter *bc;
+
+ BUG_ON(page_bc(page) != NULL);
+
+ bc = get_exec_bc();
+
+ if (bc_charge(bc, BC_KMEMSIZE, PAGE_SIZE << order,
+    (flags & __GFP_BC_LIMIT ? BC_LIMIT : BC_BARRIER)))
+  return -ENOMEM;
+
+ page_bc(page) = get_beancounter(bc);
+ return 0;
+}
+
+void bc_page_uncharge(struct page *page, int order)
+{
+ struct beancounter *bc;
+
+ bc = page_bc(page);
+ if (bc == NULL)
+  return;
+
+ bc_uncharge(bc, BC_KMEMSIZE, PAGE_SIZE << order);
+ put_beancounter(bc);
+ page_bc(page) = NULL;
+}
--- ./mm/mempool.c.bckmem 2006-04-21 11:59:36.000000000 +0400
+++ ./mm/mempool.c 2006-08-28 12:59:28.000000000 +0400
@@ -119,6 +119,7 @@ int mempool_resize(mempool_t *pool, int
  unsigned long flags;

  BUG_ON(new_min_nr <= 0);
+ gfp_mask &= ~__GFP_BC;

  spin_lock_irqsave(&pool->lock, flags);
  if (new_min_nr <= pool->min_nr) {
```

```
@@ -212,6 +213,7 @@ void * mempool_alloc(mempool_t *pool, gf
  gfp_mask |= __GFP_NOMEMALLOC; /* don't allocate emergency reserves */
  gfp_mask |= __GFP_NORETRY; /* don't loop in __alloc_pages */
  gfp_mask |= __GFP_NOWARN; /* failures are OK */
+ gfp_mask &= ~__GFP_BC;  /* do not charge */

  gfp_temp = gfp_mask & ~(__GFP_WAIT|__GFP_IO);

--- ./mm/page_alloc.c.bckmem 2006-08-28 12:20:13.000000000 +0400
+++ ./mm/page_alloc.c 2006-08-28 12:59:28.000000000 +0400
@@ -40,6 +40,8 @@
 #include <linux/sort.h>
 #include <linux/pfn.h>

+#include <bc/kmem.h>
+
 #include <asm/tlbflush.h>
 #include <asm/div64.h>
 #include "internal.h"
@@ -516,6 +518,8 @@ static void __free_pages_ok(struct page
  if (reserved)
   return;

+ bc_page_uncharge(page, order);
+
  kernel_map_pages(page, 1 << order, 0);
  local_irq_save(flags);
  __count_vm_events(PGFREE, 1 << order);
@@ -799,6 +803,8 @@ static void fastcall free_hot_cold_page(
  if (free_pages_check(page))
   return;

+ bc_page_uncharge(page, 0);
+
  kernel_map_pages(page, 1, 0);

  pcp = &zone_pcp(zone, get_cpu())->pcp[cold];
@@ -1188,6 +1194,11 @@ nopage:
   show_mem();
  }
 got_pg:
+ if ((gfp_mask & __GFP_BC) &&
+    bc_page_charge(page, order, gfp_mask)) {
+  __free_pages(page, order);
+  page = NULL;
+ }
 #ifdef CONFIG_PAGE_OWNER
  if (page)
```

```
   set_page_owner(page, order, gfp_mask);
--- ./mm/slab.c.bckmem 2006-08-28 12:20:13.000000000 +0400
+++ ./mm/slab.c 2006-08-28 12:59:28.000000000 +0400
@@ -108,6 +108,8 @@
 #include <linux/mutex.h>
 #include <linux/rtmutex.h>

+#include <bc/kmem.h>
+
 #include <asm/uaccess.h>
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>
@@ -175,11 +177,13 @@
     SLAB_CACHE_DMA | \
     SLAB_MUST_HWCACHE_ALIGN | SLAB_STORE_USER | \
     SLAB_RECLAIM_ACCOUNT | SLAB_PANIC | \
+    SLAB_BC | SLAB_BC_NOCHARGE | \
     SLAB_DESTROY_BY_RCU | SLAB_MEM_SPREAD)
 #else
 # define CREATE_MASK (SLAB_HWCACHE_ALIGN | \
     SLAB_CACHE_DMA | SLAB_MUST_HWCACHE_ALIGN | \
     SLAB_RECLAIM_ACCOUNT | SLAB_PANIC | \
+    SLAB_BC | SLAB_BC_NOCHARGE | \
     SLAB_DESTROY_BY_RCU | SLAB_MEM_SPREAD)
 #endif

@@ -793,9 +797,33 @@ static struct kmem_cache *kmem_find_gene
  return __find_general_cachep(size, gfpflags);
 }

-static size_t slab_mgmt_size(size_t nr_objs, size_t align)
+static size_t slab_mgmt_size_raw(size_t nr_objs)
 {
- return ALIGN(sizeof(struct slab)+nr_objs*sizeof(kmem_bufctl_t), align);
+ return sizeof(struct slab) + nr_objs * sizeof(kmem_bufctl_t);
+}
+
+#ifdef CONFIG_BEANCOUNTERS
+#define BC_EXTRASIZE sizeof(struct beancounter *)
+static inline size_t slab_mgmt_size_noalign(int flags, size_t nr_objs)
+{
+ size_t size;
+
+ size = slab_mgmt_size_raw(nr_objs);
+ if (flags & SLAB_BC)
+  size = ALIGN(size, BC_EXTRASIZE) + nr_objs * BC_EXTRASIZE;
+ return size;
+}
```

```
+#else
+#define BC_EXTRASIZE 0
+static inline size_t slab_mgmt_size_noalign(int flags, size_t nr_objs)
+{
+ return slab_mgmt_size_raw(nr_objs);
+}
+#endif
+
+static inline size_t slab_mgmt_size(int flags, size_t nr_objs, size_t align)
+{
+ return ALIGN(slab_mgmt_size_noalign(flags, nr_objs), align);
 }

 /*
@@ -840,20 +868,21 @@ static void cache_estimate(unsigned long
    * into account.
    */
   nr_objs = (slab_size - sizeof(struct slab)) /
-     (buffer_size + sizeof(kmem_bufctl_t));
+     (buffer_size + sizeof(kmem_bufctl_t) +
+     (flags & SLAB_BC ? BC_EXTRASIZE : 0));

   /*
    * This calculated number will be either the right
    * amount, or one greater than what we want.
    */
-  if (slab_mgmt_size(nr_objs, align) + nr_objs*buffer_size
+  if (slab_mgmt_size(flags, nr_objs, align) + nr_objs*buffer_size
          > slab_size)
    nr_objs--;

   if (nr_objs > SLAB_LIMIT)
    nr_objs = SLAB_LIMIT;

-  mgmt_size = slab_mgmt_size(nr_objs, align);
+  mgmt_size = slab_mgmt_size(flags, nr_objs, align);
  }
  *num = nr_objs;
  *left_over = slab_size - nr_objs*buffer_size - mgmt_size;
@@ -1412,7 +1441,8 @@ void __init kmem_cache_init(void)
  sizes[INDEX_AC].cs_cachep = kmem_cache_create(names[INDEX_AC].name,
      sizes[INDEX_AC].cs_size,
      ARCH_KMALLOC_MINALIGN,
-     ARCH_KMALLOC_FLAGS|SLAB_PANIC,
+     ARCH_KMALLOC_FLAGS | SLAB_BC |
+      SLAB_BC_NOCHARGE | SLAB_PANIC,
      NULL, NULL);
```

```
  if (INDEX_AC != INDEX_L3) {
@@ -1420,7 +1450,8 @@ void __init kmem_cache_init(void)
     kmem_cache_create(names[INDEX_L3].name,
      sizes[INDEX_L3].cs_size,
      ARCH_KMALLOC_MINALIGN,
-     ARCH_KMALLOC_FLAGS|SLAB_PANIC,
+     ARCH_KMALLOC_FLAGS | SLAB_BC |
+      SLAB_BC_NOCHARGE | SLAB_PANIC,
      NULL, NULL);
   }

@@ -1438,7 +1469,8 @@ void __init kmem_cache_init(void)
     sizes->cs_cachep = kmem_cache_create(names->name,
       sizes->cs_size,
       ARCH_KMALLOC_MINALIGN,
-      ARCH_KMALLOC_FLAGS|SLAB_PANIC,
+      ARCH_KMALLOC_FLAGS | SLAB_BC |
+       SLAB_BC_NOCHARGE | SLAB_PANIC,
       NULL, NULL);
    }

@@ -1941,7 +1973,8 @@ static size_t calculate_slab_order(struc
      * looping condition in cache_grow().
      */
     offslab_limit = size - sizeof(struct slab);
-    offslab_limit /= sizeof(kmem_bufctl_t);
+    offslab_limit /= (sizeof(kmem_bufctl_t) +
+      (flags & SLAB_BC ? BC_EXTRASIZE : 0));

     if (num > offslab_limit)
     break;
@@ -2249,8 +2282,8 @@ kmem_cache_create (const char *name, siz
   cachep = NULL;
   goto oops;
  }
- slab_size = ALIGN(cachep->num * sizeof(kmem_bufctl_t)
-     + sizeof(struct slab), align);
+
+ slab_size = slab_mgmt_size(flags, cachep->num, align);

  /*
   * If the slab has been placed off-slab, and we have enough space then
@@ -2261,11 +2294,9 @@ kmem_cache_create (const char *name, siz
   left_over -= slab_size;
  }

- if (flags & CFLGS_OFF_SLAB) {
+ if (flags & CFLGS_OFF_SLAB)
```

```
   /* really off slab. No need for manual alignment */
-  slab_size =
-      cachep->num * sizeof(kmem_bufctl_t) + sizeof(struct slab);
- }
+  slab_size = slab_mgmt_size_noalign(flags, cachep->num);

  cachep->colour_off = cache_line_size();
  /* Offset must be a multiple of the alignment. */
@@ -2509,6 +2540,30 @@ void kmem_cache_destroy(struct kmem_cach
 }
 EXPORT_SYMBOL(kmem_cache_destroy);

+static inline kmem_bufctl_t *slab_bufctl(struct slab *slabp)
+{
+ return (kmem_bufctl_t *) (slabp + 1);
+}
+
+#ifdef CONFIG_BEANCOUNTERS
+static inline struct beancounter **slab_bc_ptrs(kmem_cache_t *cachep,
+  struct slab *slabp)
+{
+ return (struct beancounter **) ALIGN((unsigned long)
+   (slab_bufctl(slabp) + cachep->num), BC_EXTRASIZE);
+}
+
+struct beancounter **kmem_cache_bcp(kmem_cache_t *cachep, void *objp)
+{
+ struct slab *slabp;
+ struct beancounter **bcs;
+
+ slabp = virt_to_slab(objp);
+ bcs = slab_bc_ptrs(cachep, slabp);
+ return bcs + obj_to_index(cachep, slabp, objp);
+}
+#endif
+
 /*
  * Get the memory for a slab management obj.
  * For a slab cache when the slab descriptor is off-slab, slab descriptors
@@ -2529,7 +2584,8 @@ static struct slab *alloc_slabmgmt(struc
  if (OFF_SLAB(cachep)) {
   /* Slab management obj is off-slab. */
   slabp = kmem_cache_alloc_node(cachep->slabp_cache,
-          local_flags, nodeid);
+          local_flags & (~__GFP_BC),
+          nodeid);
   if (!slabp)
    return NULL;
```

```
  } else {
@@ -2540,14 +2596,14 @@ static struct slab *alloc_slabmgmt(struc
   slabp->colouroff = colour_off;
   slabp->s_mem = objp + colour_off;
   slabp->nodeid = nodeid;
+#ifdef CONFIG_BEANCOUNTERS
+ if (cachep->flags & SLAB_BC)
+  memset(slab_bc_ptrs(cachep, slabp), 0,
+    cachep->num * BC_EXTRASIZE);
+#endif
   return slabp;
  }

-static inline kmem_bufctl_t *slab_bufctl(struct slab *slabp)
-{
- return (kmem_bufctl_t *) (slabp + 1);
-}
-
 static void cache_init_objs(struct kmem_cache *cachep,
         struct slab *slabp, unsigned long ctor_flags)
 {
@@ -2725,7 +2781,7 @@ static int cache_grow(struct kmem_cache
    * Get mem for the objs.  Attempt to allocate a physical page from
    * 'nodeid'.
    */
- objp = kmem_getpages(cachep, flags, nodeid);
+ objp = kmem_getpages(cachep, flags & (~__GFP_BC), nodeid);
   if (!objp)
    goto failed;

@@ -3073,6 +3129,19 @@ static inline void *____cache_alloc(stru
   return objp;
  }

+static inline int bc_should_charge(kmem_cache_t *cachep, gfp_t flags)
+{
+#ifdef CONFIG_BEANCOUNTERS
+ if (!(cachep->flags & SLAB_BC))
+  return 0;
+ if (flags & __GFP_BC)
+  return 1;
+ if (!(cachep->flags & SLAB_BC_NOCHARGE))
+  return 1;
+#endif
+ return 0;
+}
+
 static __always_inline void *__cache_alloc(struct kmem_cache *cachep,
```

```
       gfp_t flags, void *caller)
 {
@@ -3086,6 +3155,12 @@ static __always_inline void *__cache_all
  local_irq_restore(save_flags);
  objp = cache_alloc_debugcheck_after(cachep, flags, objp,
         caller);
+
+ if (objp && bc_should_charge(cachep, flags))
+  if (bc_slab_charge(cachep, objp, flags)) {
+   kmem_cache_free(cachep, objp);
+   objp = NULL;
+  }
  prefetchw(objp);
  return objp;
 }
@@ -3283,6 +3358,8 @@ static inline void __cache_free(struct k
  struct array_cache *ac = cpu_cache_get(cachep);

  check_irq_off();
+ if (cachep->flags & SLAB_BC)
+  bc_slab_uncharge(cachep, objp);
  objp = cache_free_debugcheck(cachep, objp, __builtin_return_address(0));

  if (cache_free_alien(cachep, objp))
--- ./mm/vmalloc.c.bckmem 2006-08-28 12:20:13.000000000 +0400
+++ ./mm/vmalloc.c 2006-08-28 12:59:28.000000000 +0400
@@ -520,6 +520,12 @@ void *vmalloc(unsigned long size)
 }
 EXPORT_SYMBOL(vmalloc);

+void *vmalloc_bc(unsigned long size)
+{
+ return __vmalloc(size, GFP_KERNEL_BC | __GFP_HIGHMEM, PAGE_KERNEL);
+}
+EXPORT_SYMBOL(vmalloc_bc);
+
 /**
  * vmalloc_user  -  allocate virtually contiguous memory which has
  *      been zeroed so it can be mapped to userspace without
```

Subject: [PATCH 7/7] BC: kernel memory (marks)
Posted by dev on Tue, 29 Aug 2006 14:56:01 GMT

View Forum Message <> Reply to Message

Mark some kmem caches with SLAB_BC and some allocations
with __GFP_BC to cause charging/limiting of appropriate
kernel resources.

Signed-off-by: Pavel Emelianov <xemul@sw.ru>
Signed-off-by: Kirill Korotaev <dev@sw.ru>

```
 arch/i386/kernel/ldt.c          |    4 ++--
 arch/i386/mm/init.c             |    4 ++--
 arch/i386/mm/pgtable.c          |    6 ++++--
 drivers/char/tty_io.c           |   10 +++++-----
 fs/file.c                       |    8 ++++----
 fs/locks.c                      |    2 +-
 fs/namespace.c                  |    3 ++-
 fs/select.c                     |    7 ++++---
 include/asm-i386/thread_info.h  |    4 ++--
 include/asm-ia64/pgalloc.h      |   24 ++++++++++++++++++-------
 include/asm-x86_64/pgalloc.h    |   12 ++++++++----
 include/asm-x86_64/thread_info.h |    5 +++--
 ipc/msgutil.c                   |    4 ++--
 ipc/sem.c                       |    7 ++++---
 ipc/util.c                      |    8 ++++----
 kernel/fork.c                   |   15 ++++++++-------
 kernel/posix-timers.c           |    3 ++-
 kernel/signal.c                 |    2 +-
 kernel/user.c                   |    2 +-
 mm/rmap.c                       |    3 ++-
 mm/shmem.c                      |    3 ++-
 21 files changed, 80 insertions(+), 56 deletions(-)

--- ./arch/i386/kernel/ldt.c.bckmemc 2006-08-28 12:20:10.000000000 +0400
+++ ./arch/i386/kernel/ldt.c 2006-08-28 13:05:46.000000000 +0400
@@ -39,9 +39,9 @@ static int alloc_ldt(mm_context_t *pc, i
  oldsize = pc->size;
  mincount = (mincount+511)&(~511);
  if (mincount*LDT_ENTRY_SIZE > PAGE_SIZE)
-  newldt = vmalloc(mincount*LDT_ENTRY_SIZE);
+  newldt = vmalloc_bc(mincount*LDT_ENTRY_SIZE);
  else
-  newldt = kmalloc(mincount*LDT_ENTRY_SIZE, GFP_KERNEL);
+  newldt = kmalloc(mincount*LDT_ENTRY_SIZE, GFP_KERNEL_BC);

  if (!newldt)
   return -ENOMEM;
--- ./arch/i386/mm/init.c.bckmemc 2006-08-28 12:20:10.000000000 +0400
+++ ./arch/i386/mm/init.c 2006-08-28 13:05:46.000000000 +0400
@@ -709,7 +709,7 @@ void __init pgtable_cache_init(void)
   pmd_cache = kmem_cache_create("pmd",
      PTRS_PER_PMD*sizeof(pmd_t),
```

```
      PTRS_PER_PMD*sizeof(pmd_t),
-     0,
+     SLAB_BC,
      pmd_ctor,
      NULL);
   if (!pmd_cache)
@@ -718,7 +718,7 @@ void __init pgtable_cache_init(void)
  pgd_cache = kmem_cache_create("pgd",
     PTRS_PER_PGD*sizeof(pgd_t),
     PTRS_PER_PGD*sizeof(pgd_t),
-    0,
+    SLAB_BC,
     pgd_ctor,
     PTRS_PER_PMD == 1 ? pgd_dtor : NULL);
  if (!pgd_cache)
--- ./arch/i386/mm/pgtable.c.bckmemc 2006-08-28 12:20:10.000000000 +0400
+++ ./arch/i386/mm/pgtable.c 2006-08-28 13:05:46.000000000 +0400
@@ -186,9 +186,11 @@ struct page *pte_alloc_one(struct mm_str
  struct page *pte;

 #ifdef CONFIG_HIGHPTE
- pte =  alloc_pages(GFP_KERNEL|__GFP_HIGHMEM|__GFP_REPEAT|__GFP_ZERO , 0);
+ pte =  alloc_pages(GFP_KERNEL|__GFP_HIGHMEM|__GFP_REPEAT|__GFP_ZERO |
+    __GFP_BC | __GFP_BC_LIMIT, 0);
 #else
- pte = alloc_pages(GFP_KERNEL|__GFP_REPEAT|__GFP_ZERO, 0);
+ pte = alloc_pages(GFP_KERNEL|__GFP_REPEAT|__GFP_ZERO|
+    __GFP_BC | __GFP_BC_LIMIT, 0);
 #endif
  return pte;
 }
--- ./drivers/char/tty_io.c.bckmemc 2006-08-28 12:20:11.000000000 +0400
+++ ./drivers/char/tty_io.c 2006-08-28 13:05:46.000000000 +0400
@@ -165,7 +165,7 @@ static void release_mem(struct tty_struc

 static struct tty_struct *alloc_tty_struct(void)
 {
- return kzalloc(sizeof(struct tty_struct), GFP_KERNEL);
+ return kzalloc(sizeof(struct tty_struct), GFP_KERNEL_BC);
 }

 static void tty_buffer_free_all(struct tty_struct *);
@@ -1904,7 +1904,7 @@ static int init_dev(struct tty_driver *d

  if (!*tp_loc) {
   tp = (struct termios *) kmalloc(sizeof(struct termios),
-      GFP_KERNEL);
+      GFP_KERNEL_BC);
```

```
   if (!tp)
    goto free_mem_out;
   *tp = driver->init_termios;
@@ -1912,7 +1912,7 @@ static int init_dev(struct tty_driver *d

  if (!*ltp_loc) {
   ltp = (struct termios *) kmalloc(sizeof(struct termios),
-       GFP_KERNEL);
+       GFP_KERNEL_BC);
   if (!ltp)
    goto free_mem_out;
   memset(ltp, 0, sizeof(struct termios));
@@ -1937,7 +1937,7 @@ static int init_dev(struct tty_driver *d

   if (!*o_tp_loc) {
    o_tp = (struct termios *)
-    kmalloc(sizeof(struct termios), GFP_KERNEL);
+    kmalloc(sizeof(struct termios), GFP_KERNEL_BC);
    if (!o_tp)
     goto free_mem_out;
    *o_tp = driver->other->init_termios;
@@ -1945,7 +1945,7 @@ static int init_dev(struct tty_driver *d

   if (!*o_ltp_loc) {
    o_ltp = (struct termios *)
-    kmalloc(sizeof(struct termios), GFP_KERNEL);
+    kmalloc(sizeof(struct termios), GFP_KERNEL_BC);
    if (!o_ltp)
     goto free_mem_out;
    memset(o_ltp, 0, sizeof(struct termios));
--- ./fs/file.c.bckmemc 2006-08-28 12:20:12.000000000 +0400
+++ ./fs/file.c 2006-08-28 13:05:46.000000000 +0400
@@ -44,9 +44,9 @@ struct file ** alloc_fd_array(int num)
  int size = num * sizeof(struct file *);

  if (size <= PAGE_SIZE)
-  new_fds = (struct file **) kmalloc(size, GFP_KERNEL);
+  new_fds = (struct file **) kmalloc(size, GFP_KERNEL_BC);
  else
-  new_fds = (struct file **) vmalloc(size);
+  new_fds = (struct file **) vmalloc_bc(size);
  return new_fds;
 }

@@ -213,9 +213,9 @@ fd_set * alloc_fdset(int num)
  int size = num / 8;

  if (size <= PAGE_SIZE)
```

```
-  new_fdset = (fd_set *) kmalloc(size, GFP_KERNEL);
+  new_fdset = (fd_set *) kmalloc(size, GFP_KERNEL_BC);
   else
-  new_fdset = (fd_set *) vmalloc(size);
+  new_fdset = (fd_set *) vmalloc_bc(size);
   return new_fdset;
  }

--- ./fs/locks.c.bckmemc 2006-08-28 12:20:12.000000000 +0400
+++ ./fs/locks.c 2006-08-28 13:05:46.000000000 +0400
@@ -2228,7 +2228,7 @@ EXPORT_SYMBOL(lock_may_write);
 static int __init filelock_init(void)
 {
  filelock_cache = kmem_cache_create("file_lock_cache",
-   sizeof(struct file_lock), 0, SLAB_PANIC,
+   sizeof(struct file_lock), 0, SLAB_PANIC | SLAB_BC,
    init_once, NULL);
  return 0;
 }
--- ./fs/namespace.c.bckmemc 2006-08-28 12:20:12.000000000 +0400
+++ ./fs/namespace.c 2006-08-28 13:05:46.000000000 +0400
@@ -1813,7 +1813,8 @@ void __init mnt_init(unsigned long mempa
  init_rwsem(&namespace_sem);

  mnt_cache = kmem_cache_create("mnt_cache", sizeof(struct vfsmount),
-   0, SLAB_HWCACHE_ALIGN | SLAB_PANIC, NULL, NULL);
+   0, SLAB_HWCACHE_ALIGN | SLAB_BC | SLAB_PANIC,
+   NULL, NULL);

  mount_hashtable = (struct list_head *)__get_free_page(GFP_ATOMIC);

--- ./fs/select.c.bckmemc 2006-08-28 12:20:12.000000000 +0400
+++ ./fs/select.c 2006-08-28 13:05:46.000000000 +0400
@@ -103,7 +103,8 @@ static struct poll_table_entry *poll_get
  if (!table || POLL_TABLE_FULL(table)) {
   struct poll_table_page *new_table;

-  new_table = (struct poll_table_page *) __get_free_page(GFP_KERNEL);
+  new_table = (struct poll_table_page *)
+   __get_free_page(GFP_KERNEL_BC);
   if (!new_table) {
    p->error = -ENOMEM;
    __set_current_state(TASK_RUNNING);
@@ -339,7 +340,7 @@ static int core_sys_select(int n, fd_set
  if (size > sizeof(stack_fds) / 6) {
   /* Not enough space in on-stack array; must use kmalloc */
   ret = -ENOMEM;
-  bits = kmalloc(6 * size, GFP_KERNEL);
```

```
+  bits = kmalloc(6 * size, GFP_KERNEL_BC);
   if (!bits)
    goto out_nofds;
  }
@@ -693,7 +694,7 @@ int do_sys_poll(struct pollfd __user *uf
   if (!stack_pp)
    stack_pp = pp = (struct poll_list *)stack_pps;
   else {
-   pp = kmalloc(size, GFP_KERNEL);
+   pp = kmalloc(size, GFP_KERNEL_BC);
    if (!pp)
     goto out_fds;
   }
--- ./include/asm-i386/thread_info.h.bckmemc 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-i386/thread_info.h 2006-08-28 13:05:46.000000000 +0400
@@ -99,13 +99,13 @@ static inline struct thread_info *curren
  ({       \
   struct thread_info *ret;   \
         \
-  ret = kmalloc(THREAD_SIZE, GFP_KERNEL);  \
+  ret = kmalloc(THREAD_SIZE, GFP_KERNEL_BC); \
   if (ret)     \
    memset(ret, 0, THREAD_SIZE);  \
   ret;      \
  })
 #else
-#define alloc_thread_info(tsk) kmalloc(THREAD_SIZE, GFP_KERNEL)
+#define alloc_thread_info(tsk) kmalloc(THREAD_SIZE, GFP_KERNEL_BC)
 #endif

 #define free_thread_info(info) kfree(info)
--- ./include/asm-ia64/pgalloc.h.bckmemc 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-ia64/pgalloc.h 2006-08-28 13:05:46.000000000 +0400
@@ -19,6 +19,8 @@
 #include <linux/page-flags.h>
 #include <linux/threads.h>

+#include <bc/kmem.h>
+
 #include <asm/mmu_context.h>

 DECLARE_PER_CPU(unsigned long *, __pgtable_quicklist);
@@ -37,7 +39,7 @@ static inline long pgtable_quicklist_tot
  return ql_size;
 }

-static inline void *pgtable_quicklist_alloc(void)
+static inline void *pgtable_quicklist_alloc(int charge)
```

```
 {
  unsigned long *ret = NULL;

@@ -45,13 +47,20 @@ static inline void *pgtable_quicklist_al

  ret = pgtable_quicklist;
  if (likely(ret != NULL)) {
+  if (charge && bc_page_charge(virt_to_page(ret),
+      0, __GFP_BC_LIMIT)) {
+   ret = NULL;
+   goto out;
+  }
   pgtable_quicklist = (unsigned long *)(*ret);
   ret[0] = 0;
   --pgtable_quicklist_size;
+out:
   preempt_enable();
  } else {
   preempt_enable();
-  ret = (unsigned long *)__get_free_page(GFP_KERNEL | __GFP_ZERO);
+  ret = (unsigned long *)__get_free_page(GFP_KERNEL |
+     __GFP_ZERO | __GFP_BC | __GFP_BC_LIMIT);
  }

  return ret;
@@ -69,6 +78,7 @@ static inline void pgtable_quicklist_fre
 #endif

  preempt_disable();
+ bc_page_uncharge(virt_to_page(pgtable_entry), 0);
  *(unsigned long *)pgtable_entry = (unsigned long)pgtable_quicklist;
  pgtable_quicklist = (unsigned long *)pgtable_entry;
  ++pgtable_quicklist_size;
@@ -77,7 +87,7 @@ static inline void pgtable_quicklist_fre

 static inline pgd_t *pgd_alloc(struct mm_struct *mm)
 {
- return pgtable_quicklist_alloc();
+ return pgtable_quicklist_alloc(1);
 }

 static inline void pgd_free(pgd_t * pgd)
@@ -94,7 +104,7 @@ pgd_populate(struct mm_struct *mm, pgd_t

 static inline pud_t *pud_alloc_one(struct mm_struct *mm, unsigned long addr)
 {
- return pgtable_quicklist_alloc();
+ return pgtable_quicklist_alloc(1);
```

```
 }

 static inline void pud_free(pud_t * pud)
@@ -112,7 +122,7 @@ pud_populate(struct mm_struct *mm, pud_t

 static inline pmd_t *pmd_alloc_one(struct mm_struct *mm, unsigned long addr)
 {
- return pgtable_quicklist_alloc();
+ return pgtable_quicklist_alloc(1);
 }

 static inline void pmd_free(pmd_t * pmd)
@@ -137,13 +147,13 @@ pmd_populate_kernel(struct mm_struct *mm
 static inline struct page *pte_alloc_one(struct mm_struct *mm,
       unsigned long addr)
 {
- return virt_to_page(pgtable_quicklist_alloc());
+ return virt_to_page(pgtable_quicklist_alloc(1));
 }

 static inline pte_t *pte_alloc_one_kernel(struct mm_struct *mm,
       unsigned long addr)
 {
- return pgtable_quicklist_alloc();
+ return pgtable_quicklist_alloc(0);
 }

 static inline void pte_free(struct page *pte)
--- ./include/asm-x86_64/pgalloc.h.bckmemc 2006-04-21 11:59:36.000000000 +0400
+++ ./include/asm-x86_64/pgalloc.h 2006-08-28 13:05:46.000000000 +0400
@@ -31,12 +31,14 @@ static inline void pmd_free(pmd_t *pmd)

 static inline pmd_t *pmd_alloc_one (struct mm_struct *mm, unsigned long addr)
 {
- return (pmd_t *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT);
+ return (pmd_t *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT|
+    __GFP_BC | __GFP_BC_LIMIT);
 }

 static inline pud_t *pud_alloc_one(struct mm_struct *mm, unsigned long addr)
 {
- return (pud_t *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT);
+ return (pud_t *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT|
+    __GFP_BC | __GFP_BC_LIMIT);
 }

 static inline void pud_free (pud_t *pud)
@@ -74,7 +76,8 @@ static inline void pgd_list_del(pgd_t *p
```

```
 static inline pgd_t *pgd_alloc(struct mm_struct *mm)
 {
  unsigned boundary;
- pgd_t *pgd = (pgd_t *)__get_free_page(GFP_KERNEL|__GFP_REPEAT);
+ pgd_t *pgd = (pgd_t *)__get_free_page(GFP_KERNEL|__GFP_REPEAT|
+    __GFP_BC | __GFP_BC_LIMIT);
  if (!pgd)
   return NULL;
  pgd_list_add(pgd);
@@ -105,7 +108,8 @@ static inline pte_t *pte_alloc_one_kerne

 static inline struct page *pte_alloc_one(struct mm_struct *mm, unsigned long address)
 {
- void *p = (void *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT);
+ void *p = (void *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT|
+    __GFP_BC | __GFP_BC_LIMIT);
  if (!p)
   return NULL;
  return virt_to_page(p);
--- ./include/asm-x86_64/thread_info.h.bckmemc 2006-08-28 12:20:13.000000000 +0400
+++ ./include/asm-x86_64/thread_info.h 2006-08-28 13:05:46.000000000 +0400
@@ -78,14 +78,15 @@ static inline struct thread_info *stack_
     ({       \
  struct thread_info *ret;    \
         \
- ret = ((struct thread_info *) __get_free_pages(GFP_KERNEL,THREAD_ORDER)); \
+ ret = ((struct thread_info *) __get_free_pages(GFP_KERNEL_BC, \
+    THREAD_ORDER));   \
  if (ret)      \
   memset(ret, 0, THREAD_SIZE);   \
  ret;       \
     })
 #else
 #define alloc_thread_info(tsk) \
- ((struct thread_info *) __get_free_pages(GFP_KERNEL,THREAD_ORDER))
+ ((struct thread_info *) __get_free_pages(GFP_KERNEL_BC,THREAD_ORDER))
 #endif

 #define free_thread_info(ti) free_pages((unsigned long) (ti), THREAD_ORDER)
--- ./ipc/msgutil.c.bckmemc 2006-04-21 11:59:36.000000000 +0400
+++ ./ipc/msgutil.c 2006-08-28 13:05:46.000000000 +0400
@@ -36,7 +36,7 @@ struct msg_msg *load_msg(const void __us
  if (alen > DATALEN_MSG)
   alen = DATALEN_MSG;

- msg = (struct msg_msg *)kmalloc(sizeof(*msg) + alen, GFP_KERNEL);
+ msg = (struct msg_msg *)kmalloc(sizeof(*msg) + alen, GFP_KERNEL_BC);
  if (msg == NULL)
```

```
   return ERR_PTR(-ENOMEM);

@@ -57,7 +57,7 @@ struct msg_msg *load_msg(const void __us
   if (alen > DATALEN_SEG)
    alen = DATALEN_SEG;
   seg = (struct msg_msgseg *)kmalloc(sizeof(*seg) + alen,
-       GFP_KERNEL);
+       GFP_KERNEL_BC);
   if (seg == NULL) {
    err = -ENOMEM;
    goto out_err;
--- ./ipc/sem.c.bckmemc 2006-08-28 12:20:13.000000000 +0400
+++ ./ipc/sem.c 2006-08-28 13:05:46.000000000 +0400
@@ -1006,7 +1006,7 @@ static inline int get_undo_list(struct s

  undo_list = current->sysvsem.undo_list;
  if (!undo_list) {
-  undo_list = kzalloc(sizeof(*undo_list), GFP_KERNEL);
+  undo_list = kzalloc(sizeof(*undo_list), GFP_KERNEL_BC);
   if (undo_list == NULL)
    return -ENOMEM;
   spin_lock_init(&undo_list->lock);
@@ -1069,7 +1069,8 @@ static struct sem_undo *find_undo(struct
  ipc_rcu_getref(sma);
  sem_unlock(sma);

- new = (struct sem_undo *) kmalloc(sizeof(struct sem_undo) + sizeof(short)*nsems,
GFP_KERNEL);
+ new = (struct sem_undo *) kmalloc(sizeof(struct sem_undo) +
+   sizeof(short)*nsems, GFP_KERNEL_BC);
  if (!new) {
   ipc_lock_by_ptr(&sma->sem_perm);
   ipc_rcu_putref(sma);
@@ -1130,7 +1131,7 @@ asmlinkage long sys_semtimedop(int semid
  if (nsops > ns->sc_semopm)
   return -E2BIG;
  if(nsops > SEMOPM_FAST) {
-  sops = kmalloc(sizeof(*sops)*nsops,GFP_KERNEL);
+  sops = kmalloc(sizeof(*sops)*nsops,GFP_KERNEL_BC);
   if(sops==NULL)
    return -ENOMEM;
  }
--- ./ipc/util.c.bckmemc 2006-08-28 12:20:13.000000000 +0400
+++ ./ipc/util.c 2006-08-28 13:05:46.000000000 +0400
@@ -406,9 +406,9 @@ void* ipc_alloc(int size)
 {
  void* out;
  if(size > PAGE_SIZE)
```

```
-  out = vmalloc(size);
+  out = vmalloc_bc(size);
   else
-  out = kmalloc(size, GFP_KERNEL);
+  out = kmalloc(size, GFP_KERNEL_BC);
   return out;
 }

@@ -491,14 +491,14 @@ void* ipc_rcu_alloc(int size)
    * workqueue if necessary (for vmalloc).
    */
   if (rcu_use_vmalloc(size)) {
-  out = vmalloc(HDRLEN_VMALLOC + size);
+  out = vmalloc_bc(HDRLEN_VMALLOC + size);
    if (out) {
     out += HDRLEN_VMALLOC;
     container_of(out, struct ipc_rcu_hdr, data)->is_vmalloc = 1;
     container_of(out, struct ipc_rcu_hdr, data)->refcount = 1;
    }
   } else {
-  out = kmalloc(HDRLEN_KMALLOC + size, GFP_KERNEL);
+  out = kmalloc(HDRLEN_KMALLOC + size, GFP_KERNEL_BC);
    if (out) {
     out += HDRLEN_KMALLOC;
     container_of(out, struct ipc_rcu_hdr, data)->is_vmalloc = 0;
--- ./kernel/fork.c.bckmemc 2006-08-28 12:52:11.000000000 +0400
+++ ./kernel/fork.c 2006-08-28 13:05:46.000000000 +0400
@@ -142,7 +142,7 @@ void __init fork_init(unsigned long memp
   /* create a slab on which task_structs can be allocated */
   task_struct_cachep =
    kmem_cache_create("task_struct", sizeof(struct task_struct),
-   ARCH_MIN_TASKALIGN, SLAB_PANIC, NULL, NULL);
+   ARCH_MIN_TASKALIGN, SLAB_PANIC | SLAB_BC, NULL, NULL);
 #endif

   /*
@@ -1435,23 +1435,24 @@ void __init proc_caches_init(void)
 {
   sighand_cachep = kmem_cache_create("sighand_cache",
     sizeof(struct sighand_struct), 0,
-   SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_DESTROY_BY_RCU,
+   SLAB_HWCACHE_ALIGN | SLAB_PANIC | \
+    SLAB_DESTROY_BY_RCU | SLAB_BC,
     sighand_ctor, NULL);
   signal_cachep = kmem_cache_create("signal_cache",
     sizeof(struct signal_struct), 0,
-   SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+   SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);
```

```
  files_cachep = kmem_cache_create("files_cache",
    sizeof(struct files_struct), 0,
-   SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+   SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);
  fs_cachep = kmem_cache_create("fs_cache",
    sizeof(struct fs_struct), 0,
-   SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+   SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);
  vm_area_cachep = kmem_cache_create("vm_area_struct",
    sizeof(struct vm_area_struct), 0,
-   SLAB_PANIC, NULL, NULL);
+   SLAB_PANIC|SLAB_BC, NULL, NULL);
  mm_cachep = kmem_cache_create("mm_struct",
    sizeof(struct mm_struct), ARCH_MIN_MMSTRUCT_ALIGN,
-   SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+   SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);
 }


--- ./kernel/posix-timers.c.bckmemc 2006-08-28 12:20:13.000000000 +0400
+++ ./kernel/posix-timers.c 2006-08-28 13:05:46.000000000 +0400
@@ -242,7 +242,8 @@ static __init int init_posix_timers(void
  register_posix_clock(CLOCK_MONOTONIC, &clock_monotonic);

  posix_timers_cache = kmem_cache_create("posix_timers_cache",
-     sizeof (struct k_itimer), 0, 0, NULL, NULL);
+     sizeof (struct k_itimer), 0, SLAB_BC,
+     NULL, NULL);
  idr_init(&posix_timers_id);
  return 0;
 }
--- ./kernel/signal.c.bckmemc 2006-08-28 12:20:13.000000000 +0400
+++ ./kernel/signal.c 2006-08-28 13:05:46.000000000 +0400
@@ -2744,5 +2744,5 @@ void __init signals_init(void)
    kmem_cache_create("sigqueue",
       sizeof(struct sigqueue),
       __alignof__(struct sigqueue),
-      SLAB_PANIC, NULL, NULL);
+      SLAB_PANIC | SLAB_BC, NULL, NULL);
 }
--- ./kernel/user.c.bckmemc 2006-08-28 12:32:54.000000000 +0400
+++ ./kernel/user.c 2006-08-28 13:05:46.000000000 +0400
@@ -194,7 +194,7 @@ static int __init uid_cache_init(void)
  int n;

  uid_cachep = kmem_cache_create("uid_cache", sizeof(struct user_struct),
-   0, SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+   0, SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);
```

```
  for(n = 0; n < UIDHASH_SZ; ++n)
   INIT_LIST_HEAD(uidhash_table + n);
--- ./mm/rmap.c.bckmemc 2006-08-28 12:20:13.000000000 +0400
+++ ./mm/rmap.c 2006-08-28 13:05:46.000000000 +0400
@@ -179,7 +179,8 @@ static void anon_vma_ctor(void *data, st
 void __init anon_vma_init(void)
 {
  anon_vma_cachep = kmem_cache_create("anon_vma", sizeof(struct anon_vma),
-   0, SLAB_DESTROY_BY_RCU|SLAB_PANIC, anon_vma_ctor, NULL);
+   0, SLAB_DESTROY_BY_RCU|SLAB_PANIC|SLAB_BC,
+   anon_vma_ctor, NULL);
 }

 /*
--- ./mm/shmem.c.bckmemc 2006-08-28 12:20:13.000000000 +0400
+++ ./mm/shmem.c 2006-08-28 13:05:46.000000000 +0400
@@ -368,7 +368,8 @@ static swp_entry_t *shmem_swp_alloc(stru
   }

   spin_unlock(&info->lock);
-  page = shmem_dir_alloc(mapping_gfp_mask(inode->i_mapping) | __GFP_ZERO);
+  page = shmem_dir_alloc(mapping_gfp_mask(inode->i_mapping) | \
+    __GFP_ZERO | __GFP_BC);
   if (page)
    set_page_private(page, 0);
   spin_lock(&info->lock);
```

Subject: Re: [PATCH 6/7] BC: kernel memory (core)
Posted by Balbir Singh on Tue, 29 Aug 2006 18:41:46 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
> Introduce BC_KMEMSIZE resource which accounts kernel
> objects allocated by task's request.
>
> Reference to BC is kept on struct page or slab object.
> For slabs each struct slab contains a set of pointers
> corresponding objects are charged to.
>
> Allocation charge rules:
> 1. Pages - if allocation is performed with __GFP_BC flag - page
>   is charged to current's exec_bc.
> 2. Slabs - kmem_cache may be created with SLAB_BC flag - in this
>   case each allocation is charged. Caches used by kmalloc are
>   created with SLAB_BC | SLAB_BC_NOCHARGE flags. In this case
>   only __GFP_BC allocations are charged.

>

<snip>

> +#define __GFP_BC_LIMIT ((__force gfp_t)0x100000u) /* Charge against BC
> limit */
>

What's _GFP_BC_LIMIT for, could you add the description for that flag?
The comment is not very clear

> +#ifdef CONFIG_BEANCOUNTERS
> +   union {
> +   struct beancounter   *page_bc;
> +   } bc;
> +#endif
> };
>
> +#define page_bc(page)   ((page)->bc.page_bc)

Minor comment - page->(bc).page_bc has too many repititions of page and bc - see
the Practice of Programming by Kernighan and Pike

I missed the part of why you wanted to have a union (in struct page for bc)?

> const char *bc_rnames[] = {
> +   "kmemsize",   /* 0 */
> };
>
> static struct hlist_head bc_hash[BC_HASH_SIZE];
> @@ -221,6 +222,8 @@ static void init_beancounter_syslimits(s
> {
>   int k;
>
> +   bc->bc_parms[BC_KMEMSIZE].limit = 32 * 1024 * 1024;
> +

Can't this be configurable CONFIG_XXX or a #defined constant?

> --- ./mm/mempool.c.bckmem   2006-04-21 11:59:36.000000000 +0400
> +++ ./mm/mempool.c   2006-08-28 12:59:28.000000000 +0400
> @@ -119,6 +119,7 @@ int mempool_resize(mempool_t *pool, int   unsigned
> long flags;
>
>   BUG_ON(new_min_nr <= 0);
> +   gfp_mask &= ~__GFP_BC;
>
>   spin_lock_irqsave(&pool->lock, flags);

```
>     if (new_min_nr <= pool->min_nr) {
> @@ -212,6 +213,7 @@ void * mempool_alloc(mempool_t *pool, gf
>     gfp_mask |= __GFP_NOMEMALLOC;    /* don't allocate emergency
> reserves */
>     gfp_mask |= __GFP_NORETRY;    /* don't loop in __alloc_pages */
>     gfp_mask |= __GFP_NOWARN;    /* failures are OK */
> +    gfp_mask &= ~__GFP_BC;        /* do not charge */
>
>     gfp_temp = gfp_mask & ~(__GFP_WAIT|__GFP_IO);
>
```

Is there any reasn why mempool_xxxx() functions are not charged? Is it because mempool functions are mostly used from the I/O path?

```
> --- ./mm/page_alloc.c.bckmem    2006-08-28 12:20:13.000000000 +0400
> +++ ./mm/page_alloc.c    2006-08-28 12:59:28.000000000 +0400
> @@ -40,6 +40,8 @@
> #include <linux/sort.h>
> #include <linux/pfn.h>
>
> +#include <bc/kmem.h>
> +
> #include <asm/tlbflush.h>
> #include <asm/div64.h>
> #include "internal.h"
> @@ -516,6 +518,8 @@ static void __free_pages_ok(struct page     if
> (reserved)
>         return;
>
> +    bc_page_uncharge(page, order);
> +
>     kernel_map_pages(page, 1 << order, 0);
>     local_irq_save(flags);
>     __count_vm_events(PGFREE, 1 << order);
> @@ -799,6 +803,8 @@ static void fastcall free_hot_cold_page(
>     if (free_pages_check(page))
>         return;
>
> +    bc_page_uncharge(page, 0);
> +
>     kernel_map_pages(page, 1, 0);
>
>     pcp = &zone_pcp(zone, get_cpu())->pcp[cold];
> @@ -1188,6 +1194,11 @@ nopage:
>         show_mem();
>     }
> got_pg:
> +    if ((gfp_mask & __GFP_BC) &&
```

> + bc_page_charge(page, order, gfp_mask)) {

I wonder if bc_page_charge() should be called bc_page_charge_failed()?
Does it make sense to atleast partially start reclamation here? I know with
bean counters we cannot reclaim from a particular container, but for now
we could kick off kswapd() or call shrink_all_memory() inline (Dave's patches do
this to shrink memory from the particular cpuset). Or do you want to leave this
slot open for later?

> + __free_pages(page, order);
> + page = NULL;
> + }

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH 6/7] BC: kernel memory (core)
Posted by Dave Hansen on Tue, 29 Aug 2006 20:29:42 GMT
View Forum Message <> Reply to Message

On Tue, 2006-08-29 at 18:58 +0400, Kirill Korotaev wrote:
> @@ -274,8 +274,14 @@ struct page {
> unsigned int gfp_mask;
> unsigned long trace[8];
> #endif
> +#ifdef CONFIG_BEANCOUNTERS
> + union {
> + struct beancounter *page_bc;
> + } bc;
> +#endif
> };

I know you're probably saving this union for when you put some userspace
stuff in here or something. But, for now, it would probably be best
just to leave it as a plain struct, or even an anonymous union.

You probably had to use gcc 2 when this was written and couldn't use
anonymous unions, right?

-- Dave

Subject: Re: [PATCH 2/7] BC: kconfig
Posted by rdunlap on Wed, 30 Aug 2006 04:02:53 GMT
View Forum Message <> Reply to Message

On Tue, 29 Aug 2006 18:51:20 +0400 Kirill Korotaev wrote:

> Add kernel/bc/Kconfig file with BC options and
> include it into arch Kconfigs
>
> ---
> init/Kconfig | 2 ++
> kernel/bc/Kconfig | 25 +++++++++++++++++++++++++
> 2 files changed, 27 insertions(+)
>
> --- ./init/Kconfig.bckm 2006-07-10 12:39:10.000000000 +0400
> +++ ./init/Kconfig 2006-07-28 14:10:41.000000000 +0400
> @@ -222,6 +222,8 @@ source "crypto/Kconfig"
>
>   Say N if unsure.
>
> +source "kernel/bc/Kconfig"
> +
> config SYSCTL
>   bool
>
> --- ./kernel/bc/Kconfig.bckm 2006-07-28 13:07:38.000000000 +0400
> +++ ./kernel/bc/Kconfig 2006-07-28 13:09:51.000000000 +0400
> @@ -0,0 +1,25 @@
> +#
> +# Resource beancounters (BC)
> +#
> +# Copyright (C) 2006 OpenVZ. SWsoft Inc
> +
> +menu "User resources"
> +
> +config BEANCOUNTERS
> + bool "Enable resource accounting/control"
> + default n
> + help
> +   When Y this option provides accounting and allows to configure

"allows" needs an object after it, like "you" or "one".

> +   limits for user's consumption of exhaustible system resources.
> +   The most important resource controlled by this patch is unswappable
> +   memory (either mlock'ed or used by internal kernel structures and
> +   buffers). The main goal of this patch is to protect processes
> +   from running short of important resources because of an accidental

drop "an"

> +	misbehavior of processes or malicious activity aiming to ``kill"
> +	the system. It's worth to mention that resource limits configured

s/to mention/mentioning/

> +	by setrlimit(2) do not give an acceptable level of protection
> +	because they cover only small fraction of resources and work on a

"only a small fraction"

> +	per-process basis. Per-process accounting doesn't prevent malicious
> +	users from spawning a lot of resource-consuming processes.
> +
> +endmenu

and there are several lines that end with <space> (don't do that :).

~Randy

Subject: Re: [PATCH 1/7] introduce atomic_dec_and_lock_irqsave()
Posted by Roman Zippel on Wed, 30 Aug 2006 09:59:48 GMT
View Forum Message <> Reply to Message

Hi,

On Tue, 29 Aug 2006, Kirill Korotaev wrote:

> --- ./kernel/user.c.dlirq 2006-07-10 12:39:20.000000000 +0400
> +++ ./kernel/user.c 2006-08-28 11:08:56.000000000 +0400
> @@ -108,15 +108,12 @@ void free_uid(struct user_struct *up)
> if (!up)
> 	return;
>
> - local_irq_save(flags);
> - if (atomic_dec_and_lock(&up->__count, &uidhash_lock)) {
> + if (atomic_dec_and_lock_irqsave(&up->__count, &uidhash_lock, flags)) {
> 	uid_hash_remove(up);
> 	spin_unlock_irqrestore(&uidhash_lock, flags);
> 	key_put(up->uid_keyring);
> 	key_put(up->session_keyring);
> 	kmem_cache_free(uid_cachep, up);
> - } else {
> - local_irq_restore(flags);
> }

> }

Why does this need protection against interrupts?

bye, Roman

Subject: Re: [PATCH 1/7] introduce atomic_dec_and_lock_irqsave()
Posted by Roman Zippel on Wed, 30 Aug 2006 10:51:28 GMT
View Forum Message <> Reply to Message

Hi,

On Wed, 30 Aug 2006, Oleg Nesterov wrote:

> > Why does this need protection against interrupts?
>
> uidhash_lock can be taken from irq context. For example, delayed_put_task_struct()
> does __put_task_struct()->free_uid().

AFAICT it's called via rcu, does that mean anything released via rcu has
to be protected against interrupts?

bye, Roman

Subject: Re: [PATCH 1/7] introduce atomic_dec_and_lock_irqsave()
Posted by Oleg Nesterov on Wed, 30 Aug 2006 11:27:53 GMT
View Forum Message <> Reply to Message

On 08/30, Roman Zippel wrote:
> Hi,
>
> On Wed, 30 Aug 2006, Oleg Nesterov wrote:
>
> > > Why does this need protection against interrupts?
> >
> > uidhash_lock can be taken from irq context. For example, delayed_put_task_struct()
> > does __put_task_struct()->free_uid().
>
> AFAICT it's called via rcu, does that mean anything released via rcu has
> to be protected against interrupts?

RCU means softirq, probably we can use spin_lock_bh() to protect against deadlock.
But free_uid() may be called with irqs disabled, we can't use local_bh_enable()
in such a case.

Oleg.

Subject: Re: [PATCH 1/7] introduce atomic_dec_and_lock_irqsave()
Posted by Roman Zippel on Wed, 30 Aug 2006 11:54:51 GMT
View Forum Message <> Reply to Message

Hi,

On Wed, 30 Aug 2006, Oleg Nesterov wrote:

> > AFAICT it's called via rcu, does that mean anything released via rcu has
> > to be protected against interrupts?
>
> RCU means softirq, probably we can use spin_lock_bh() to protect against deadlock.
> But free_uid() may be called with irqs disabled, we can't use local_bh_enable()
> in such a case.

Eek, I wasn't really aware of it and this would really suck. We should
move things out of the interrupt context and not into it. :(
I would call it a bug in the rcu system.

bye, Roman

Subject: Re: [PATCH 1/7] introduce atomic_dec_and_lock_irqsave()
Posted by Oleg Nesterov on Wed, 30 Aug 2006 12:01:38 GMT
View Forum Message <> Reply to Message

On 08/30, Roman Zippel wrote:
> Hi,
>
> On Tue, 29 Aug 2006, Kirill Korotaev wrote:
>
> > --- ./kernel/user.c.dlirq 2006-07-10 12:39:20.000000000 +0400
> > +++ ./kernel/user.c 2006-08-28 11:08:56.000000000 +0400
> > @@ -108,15 +108,12 @@ void free_uid(struct user_struct *up)
> > if (!up)
> > return;
> >
> > - local_irq_save(flags);
> > - if (atomic_dec_and_lock(&up->__count, &uidhash_lock)) {
> > + if (atomic_dec_and_lock_irqsave(&up->__count, &uidhash_lock, flags)) {
> > uid_hash_remove(up);
> > spin_unlock_irqrestore(&uidhash_lock, flags);
> > key_put(up->uid_keyring);
> > key_put(up->session_keyring);

> >   kmem_cache_free(uid_cachep, up);
> > - } else {
> > - local_irq_restore(flags);
> > }
> > }
>
> Why does this need protection against interrupts?

uidhash_lock can be taken from irq context. For example, delayed_put_task_struct()
does __put_task_struct()->free_uid().

Oleg.

Subject: Re: [PATCH 1/7] introduce atomic_dec_and_lock_irqsave()
Posted by Dipankar Sarma on Wed, 30 Aug 2006 16:58:51 GMT
View Forum Message <> Reply to Message

On Wed, Aug 30, 2006 at 12:51:28PM +0200, Roman Zippel wrote:
> Hi,
>
> On Wed, 30 Aug 2006, Oleg Nesterov wrote:
>
> > > Why does this need protection against interrupts?
> >
> > uidhash_lock can be taken from irq context. For example, delayed_put_task_struct()
> > does __put_task_struct()->free_uid().
>
> AFAICT it's called via rcu, does that mean anything released via rcu has
> to be protected against interrupts?

No. You need protection only if you have are using some
data that can also be used by the RCU callback. For example,
if your RCU callback just calls kfree(), you don't have to
do a spin_lock_bh().

Thanks
Dipankar

Subject: Re: [PATCH 1/7] introduce atomic_dec_and_lock_irqsave()
Posted by Roman Zippel on Wed, 30 Aug 2006 17:25:07 GMT
View Forum Message <> Reply to Message

Hi,

On Wed, 30 Aug 2006, Dipankar Sarma wrote:

> > > uidhash_lock can be taken from irq context. For example, delayed_put_task_struct()
> > > does __put_task_struct()->free_uid().
> >
> > AFAICT it's called via rcu, does that mean anything released via rcu has
> > to be protected against interrupts?
>
> No. You need protection only if you have are using some
> data that can also be used by the RCU callback. For example,
> if your RCU callback just calls kfree(), you don't have to
> do a spin_lock_bh().

In this case kfree() does its own interrupt synchronization. I didn't
realize before that rcu had this (IMO serious) limitation. I think there
should be two call_rcu() variants, one that queues the callback in a soft
irq and a second which queues it in a thread context.

bye, Roman

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; /* minumum 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 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 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.

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.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH 4/7] BC: context inheriting and changing
Posted by Chandra Seetharaman on Wed, 30 Aug 2006 19:11:32 GMT
View Forum Message <> Reply to Message

On Tue, 2006-08-29 at 18:54 +0400, Kirill Korotaev wrote:
> Contains code responsible for setting BC on task,
> it's inheriting and setting host context in interrupts.
>
> Task references 2 beancounters:
> 1. exec_bc: current context. all resources are
> charged to this beancounter.
> 3. fork_bc: beancounter which is inherited by
> task's children on fork
>
> Signed-off-by: Pavel Emelianov <xemul@sw.ru>
> Signed-off-by: Kirill Korotaev <dev@sw.ru>
>
> ---
>
> include/bc/task.h | 59 ++
> include/linux/sched.h | 5 ++++
> kernel/bc/Makefile | 1
> kernel/bc/beancounter.c | 3 ++
> kernel/bc/misc.c | 32 ++++++++++++++++++++++++++++++
> kernel/fork.c | 17 ++++++++++++++-
> kernel/irq/handle.c | 9 +++++++
> kernel/softirq.c | 8 ++++++
> 8 files changed, 132 insertions(+), 2 deletions(-)
>
> --- /dev/null 2006-07-18 14:52:43.075228448 +0400
> +++ ./include/bc/task.h 2006-08-28 12:54:48.000000000 +0400
> @@ -0,0 +1,59 @@
> +/*
> + * include/bc/task.h
> + *
> + * Copyright (C) 2006 OpenVZ. SWsoft Inc

```
> + *
> + */
> +
> +#ifndef __BC_TASK_H_
> +#define __BC_TASK_H_
> +
> +struct beancounter;
```

we can as well include beancounter.h here and avoid including it in
different files below.

```
> +
> +struct task_beancounter {
> + struct beancounter *exec_bc;
> + struct beancounter *fork_bc;
> +};
> +
> +#ifdef CONFIG_BEANCOUNTERS
> +
> +#define get_exec_bc()  (current->task_bc.exec_bc)
> +#define set_exec_bc(new)   \
> + ({     \
> +   struct beancounter *old; \
> +   struct task_beancounter *tbc; \
> +   tbc = &current->task_bc; \
> +   old = tbc->exec_bc;  \
> +   tbc->exec_bc = new;  \
> +   old;    \
> +  })
> +#define reset_exec_bc(old, exp)   \
> + do {     \
> +  struct task_beancounter *tbc; \
> +  tbc = &current->task_bc; \
> +  BUG_ON(tbc->exec_bc != exp); \
> +  tbc->exec_bc = old;  \
> + } while (0)
> +
```

Didn't follow why the above two macros can't be functions.

```
> +int __must_check bc_task_charge(struct task_struct *parent,
> +  struct task_struct *new);
> +void bc_task_uncharge(struct task_struct *tsk);
> +
> +#else
> +
> +#define get_exec_bc()  (NULL)
> +#define set_exec_bc(new) (NULL)
```

```
> +#define reset_exec_bc(new, exp) do { } while (0)
> +
> +static inline __must_check int bc_task_charge(struct task_struct *parent,
> +  struct task_struct *new)
> +{
> + return 0;
> +}
> +
> +static inline void bc_task_uncharge(struct task_struct *tsk)
> +{
> +}
> +
> +#endif
> +#endif
> --- ./include/linux/sched.h.bctask 2006-08-28 12:20:13.000000000 +0400
> +++ ./include/linux/sched.h 2006-08-28 12:52:11.000000000 +0400
> @@ -83,6 +83,8 @@ struct sched_param {
>  #include <linux/timer.h>
>  #include <linux/hrtimer.h>
>
> +#include <bc/task.h>
> +
>  #include <asm/processor.h>
>
>  struct exec_domain;
> @@ -1048,6 +1050,9 @@ struct task_struct {
>   spinlock_t delays_lock;
>   struct task_delay_info *delays;
>  #endif
> +#ifdef CONFIG_BEANCOUNTERS
> + struct task_beancounter task_bc;
> +#endif
>  };
>
>  static inline pid_t process_group(struct task_struct *tsk)
> --- ./kernel/bc/Makefile.bctask 2006-08-28 12:43:34.000000000 +0400
> +++ ./kernel/bc/Makefile 2006-08-28 12:52:11.000000000 +0400
> @@ -5,3 +5,4 @@
>  #
>
>  obj-$(CONFIG_BEANCOUNTERS) += beancounter.o
> +obj-$(CONFIG_BEANCOUNTERS) += misc.o
> --- ./kernel/bc/beancounter.c.bctask 2006-08-28 12:49:07.000000000 +0400
> +++ ./kernel/bc/beancounter.c 2006-08-28 12:52:11.000000000 +0400
> @@ -238,6 +238,9 @@ void __init bc_init_early(void)
>   spin_lock_init(&bc_hash_lock);
>   slot = &bc_hash[bc_hash_fn(bc->bc_id)];
>   hlist_add_head(&bc->hash, slot);
```

> +
> + current->task_bc.exec_bc = get_beancounter(bc);
> + current->task_bc.fork_bc = get_beancounter(bc);
> }
>
> void __init bc_init_late(void)
> --- /dev/null 2006-07-18 14:52:43.075228448 +0400
> +++ ./kernel/bc/misc.c 2006-08-28 12:52:11.000000000 +0400
> @@ -0,0 +1,32 @@
> +/*
> + * kernel/bc/misc.c
> + *
> + * Copyright (C) 2006 OpenVZ. SWsoft Inc.
> + *
> + */
> +
> +#include <linux/sched.h>
> +
> +#include <bc/beancounter.h>
> +#include <bc/task.h>

task.h is included via sched.h, not needed here.
> +
> +int bc_task_charge(struct task_struct *parent, struct task_struct *new)
> +{
> + struct task_beancounter *old_bc;
> + struct task_beancounter *new_bc;
> + struct beancounter *bc;
> +
> + old_bc = &parent->task_bc;
> + new_bc = &new->task_bc;
> +
> + bc = old_bc->fork_bc;
> + new_bc->exec_bc = get_beancounter(bc);
> + new_bc->fork_bc = get_beancounter(bc);
> + return 0;
> +}

There is no failures, why not return void ?

> +
> +void bc_task_uncharge(struct task_struct *tsk)
> +{
> + put_beancounter(tsk->task_bc.exec_bc);
> + put_beancounter(tsk->task_bc.fork_bc);
> +}
> --- ./kernel/fork.c.bctask 2006-08-28 12:20:13.000000000 +0400
> +++ ./kernel/fork.c 2006-08-28 12:52:11.000000000 +0400

```
> @@ -49,6 +49,8 @@
>  #include <linux/taskstats_kern.h>
>  #include <linux/random.h>
>
> +#include <bc/task.h>
> +
```

not needed as already included in sched.h

```
>  #include <asm/pgtable.h>
>  #include <asm/pgalloc.h>
>  #include <asm/uaccess.h>
> @@ -103,12 +105,18 @@ kmem_cache_t *vm_area_cachep;
>  /* SLAB cache for mm_struct structures (tsk->mm) */
>  static kmem_cache_t *mm_cachep;
>
> -void free_task(struct task_struct *tsk)
> +static void __free_task(struct task_struct *tsk)
>  {
>   free_thread_info(tsk->thread_info);
>   rt_mutex_debug_task_free(tsk);
>   free_task_struct(tsk);
>  }
> +
> +void free_task(struct task_struct *tsk)
> +{
> + bc_task_uncharge(tsk);
> + __free_task(tsk);
> +}
>  EXPORT_SYMBOL(free_task);
>
>  void __put_task_struct(struct task_struct *tsk)
> @@ -983,6 +991,9 @@ static struct task_struct *copy_process(
>   if (!p)
>    goto fork_out;
>
> + if (bc_task_charge(current, p))
> +  goto bad_charge;
> +
>  #ifdef CONFIG_TRACE_IRQFLAGS
>   DEBUG_LOCKS_WARN_ON(!p->hardirqs_enabled);
>   DEBUG_LOCKS_WARN_ON(!p->softirqs_enabled);
> @@ -1293,7 +1304,9 @@ bad_fork_cleanup_count:
>   atomic_dec(&p->user->processes);
>   free_uid(p->user);
>  bad_fork_free:
> - free_task(p);
> + bc_task_uncharge(p);
```

> +bad_charge:
> + __free_task(p);
> fork_out:
>   return ERR_PTR(retval);
> }
> --- ./kernel/irq/handle.c.bctask 2006-07-10 12:39:20.000000000 +0400
> +++ ./kernel/irq/handle.c 2006-08-28 12:52:11.000000000 +0400
> @@ -16,6 +16,9 @@
> #include <linux/interrupt.h>
> #include <linux/kernel_stat.h>
>
> +#include <bc/beancounter.h>
> +#include <bc/task.h>

same here
> +
> #include "internals.h"
>
> /**
> @@ -166,6 +169,9 @@ fastcall unsigned int __do_IRQ(unsigned
>   struct irq_desc *desc = irq_desc + irq;
>   struct irqaction *action;
>   unsigned int status;
> + struct beancounter *bc;
> +
> + bc = set_exec_bc(&init_bc);
>
>   kstat_this_cpu.irqs[irq]++;
>   if (CHECK_IRQ_PER_CPU(desc->status)) {
> @@ -178,6 +184,8 @@ fastcall unsigned int __do_IRQ(unsigned
>   desc->chip->ack(irq);
>   action_ret = handle_IRQ_event(irq, regs, desc->action);
>   desc->chip->end(irq);
> +
> + reset_exec_bc(bc, &init_bc);
>   return 1;
>   }
>
> @@ -246,6 +254,7 @@ out:
>   desc->chip->end(irq);
>   spin_unlock(&desc->lock);
>
> + reset_exec_bc(bc, &init_bc);
>   return 1;
> }
>
> --- ./kernel/softirq.c.bctask 2006-08-28 12:20:13.000000000 +0400
> +++ ./kernel/softirq.c 2006-08-28 12:52:11.000000000 +0400

> @@ -18,6 +18,9 @@
> #include <linux/rcupdate.h>
> #include <linux/smp.h>
>
> +#include <bc/beancounter.h>
> +#include <bc/task.h>

same here

> +
> #include <asm/irq.h>
> /*
>   - No shared variables, all the data are CPU local.
> @@ -209,6 +212,9 @@ asmlinkage void __do_softirq(void)
>   __u32 pending;
>   int max_restart = MAX_SOFTIRQ_RESTART;
>   int cpu;
> + struct beancounter *bc;
> +
> + bc = set_exec_bc(&init_bc);
>
>   pending = local_softirq_pending();
>   account_system_vtime(current);
> @@ -247,6 +253,8 @@ restart:
>
>   account_system_vtime(current);
>   _local_bh_enable();
> +
> + reset_exec_bc(bc, &init_bc);
> }
>
> #ifndef __ARCH_HAS_DO_SOFTIRQ
>
> --- -------------
> 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.
 --- ----------

Subject: Re: [ckrm-tech] [PATCH 5/7] BC: user interface (syscalls)
Posted by Chandra Seetharaman on Wed, 30 Aug 2006 19:15:23 GMT
View Forum Message <> Reply to Message

On Tue, 2006-08-29 at 18:55 +0400, Kirill Korotaev wrote:
> Add the following system calls for BC management:
> 1. sys_get_bcid   - get current BC id
> 2. sys_set_bcid   - change exec_ and fork_ BCs on current
> 3. sys_set_bclimit - set limits for resources consumtions
> 4. sys_get_bcstat   - return br_resource_parm on resource
>
> Signed-off-by: Pavel Emelianov <xemul@sw.ru>
> Signed-off-by: Kirill Korotaev <dev@sw.ru>
>
> ---
<snip>

> +
> +asmlinkage long sys_set_bclimit(bcid_t id, unsigned long resource,
> + unsigned long __user *limits)
> +{
> + int error;
> + unsigned long flags;
> + struct beancounter *bc;
> + unsigned long new_limits[2];
> +
> + error = -EPERM;
> + if(!capable(CAP_SYS_RESOURCE))
> + goto out;
> +
> + error = -EINVAL;
> + if (resource >= BC_RESOURCES)
> + goto out;
> +
> + error = -EFAULT;
> + if (copy_from_user(&new_limits, limits, sizeof(new_limits)))
> + goto out;
> +
> + error = -EINVAL;
> + if (new_limits[0] > BC_MAXVALUE || new_limits[1] > BC_MAXVALUE ||
> +   new_limits[0] > new_limits[1])
> + goto out;
> +
> + error = -ENOENT;
> + bc = beancounter_findcreate(id, BC_LOOKUP);
> + if (bc == NULL)
> + goto out;

Moving this to be before copy_from_user() would be efficient.

> +
> + spin_lock_irqsave(&bc->bc_lock, flags);
> + bc->bc_parms[resource].barrier = new_limits[0];
> + bc->bc_parms[resource].limit = new_limits[1];
> + spin_unlock_irqrestore(&bc->bc_lock, flags);
> +
> + put_beancounter(bc);
> + error = 0;
> +out:
> + return error;
> +}
<snip>
--

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

Subject: Re: [ckrm-tech] [PATCH 6/7] BC: kernel memory (core)
Posted by Chandra Seetharaman on Wed, 30 Aug 2006 19:25:26 GMT
View Forum Message <> Reply to Message

On Tue, 2006-08-29 at 18:58 +0400, Kirill Korotaev wrote:
> --- ./include/bc/beancounter.h.bckmem 2006-08-28 12:47:52.000000000 +0400
> +++ ./include/bc/beancounter.h 2006-08-28 12:59:28.000000000 +0400
> @@ -12,7 +12,9 @@
> * Resource list.
> */
>
> -#define BC_RESOURCES 0
> +#define BC_KMEMSIZE 0
> +
> +#define BC_RESOURCES 1

<snip>

> --- ./kernel/bc/beancounter.c.bckmem 2006-08-28 12:52:11.000000000 +0400
> +++ ./kernel/bc/beancounter.c 2006-08-28 12:59:28.000000000 +0400
> @@ -20,6 +20,7 @@ static void init_beancounter_struct(stru
> struct beancounter init_bc;
>
> const char *bc_rnames[] = {
> + "kmemsize", /* 0 */
> };
>
> static struct hlist_head bc_hash[BC_HASH_SIZE];

> @@ -221,6 +222,8 @@ static void init_beancounter_syslimits(s
> {
> int k;
>
> + bc->bc_parms[BC_KMEMSIZE].limit = 32 * 1024 * 1024;
> +
> for (k = 0; k < BC_RESOURCES; k++)
> bc->bc_parms[k].barrier = bc->bc_parms[k].limit;
> }

As I mentioned in one of my earlier email
(http://marc.theaimsgroup.com/?l=linux-kernel&m=115619384 500289&w=2),
IMHO, this way of defining an interface is not clean/clear (for
controller writers).

<snip>
--

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

Subject: Re: [ckrm-tech] [PATCH 3/7] BC: beancounters core (API)
Posted by Andrew Morton on Wed, 30 Aug 2006 19:47:52 GMT
View Forum Message <> Reply to Message

On Wed, 30 Aug 2006 11:58:34 -0700
Chandra Seetharaman <sekharan@us.ibm.com> wrote:

> > +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).

argh. Those few characters are useful.

Subject: Re: [PATCH 1/7] introduce atomic_dec_and_lock_irqsave()
Posted by paulmck on Thu, 31 Aug 2006 22:58:28 GMT
View Forum Message <> Reply to Message

On Wed, Aug 30, 2006 at 07:25:07PM +0200, Roman Zippel wrote:

> Hi,
>
> On Wed, 30 Aug 2006, Dipankar Sarma wrote:
>
> > > > uidhash_lock can be taken from irq context. For example, delayed_put_task_struct()
> > > > does __put_task_struct()->free_uid().
> > >
> > > AFAICT it's called via rcu, does that mean anything released via rcu has
> > > to be protected against interrupts?
> >
> > No. You need protection only if you have are using some
> > data that can also be used by the RCU callback. For example,
> > if your RCU callback just calls kfree(), you don't have to
> > do a spin_lock_bh().
>
> In this case kfree() does its own interrupt synchronization. I didn't
> realize before that rcu had this (IMO serious) limitation. I think there
> should be two call_rcu() variants, one that queues the callback in a soft
> irq and a second which queues it in a thread context.

How about just using synchronize_rcu() in the second situation?
This primitive blocks until the grace period completes, allowing you to
do the remaining processing in thread context. As a bonus, RCU code
that uses synchronize_rcu() is usually quite a bit simpler than code
using call_rcu().

Using synchronize_rcu():

```
 list_del_rcu(p);
 synchronize_rcu();
 kfree(p);
```

Using call_rcu():

```
 static void rcu_callback_func(struct rcu_head *rcu)
 {
  struct foo *p = container_of(rcu, struct foo, rcu);

  kfree(p);
 }

 list_del_rcu(p);
 call_rcu(&p->rcu, rcu_callback_func);
```

Furthermore, the call_rcu() approach requires a struct rcu_head somewhere
in the data structure, so use of synchronize_rcu() saves a bit of memory,
as well.

But if you have a situation where neither synchronize_srcu() nor
call_rcu() is working out for you, let's hear it!

Thanx, Paul

Subject: Re: [PATCH 6/7] BC: kernel memory (core)
Posted by dev on Mon, 04 Sep 2006 12:19:10 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Kirill Korotaev wrote:
>
>> Introduce BC_KMEMSIZE resource which accounts kernel
>> objects allocated by task's request.
>>
>> Reference to BC is kept on struct page or slab object.
>> For slabs each struct slab contains a set of pointers
>> corresponding objects are charged to.
>>
>> Allocation charge rules:
>> 1. Pages - if allocation is performed with __GFP_BC flag - page
>>   is charged to current's exec_bc.
>> 2. Slabs - kmem_cache may be created with SLAB_BC flag - in this
>>   case each allocation is charged. Caches used by kmalloc are
>>   created with SLAB_BC | SLAB_BC_NOCHARGE flags. In this case
>>   only __GFP_BC allocations are charged.
>>
>
> <snip>
>
>> +#define __GFP_BC_LIMIT ((__force gfp_t)0x100000u) /* Charge against
>> BC limit */
>>
>
> What's _GFP_BC_LIMIT for, could you add the description for that flag?
> The comment is not very clear
>
>> +#ifdef CONFIG_BEANCOUNTERS
>> +   union {
>> +   struct beancounter   *page_bc;
>> +   } bc;
>> +#endif
>> };
>>
>> +#define page_bc(page)   ((page)->bc.page_bc)
>
>

> Minor comment - page->(bc).page_bc has too many repititions of page and
> bc - see
> the Practice of Programming by Kernighan and Pike
>
> I missed the part of why you wanted to have a union (in struct page for
> bc)?
because this union is used both for kernel memory accounting and user memeory tracking.

>> const char *bc_rnames[] = {
>> +   "kmemsize",   /* 0 */
>> };
>>
>> static struct hlist_head bc_hash[BC_HASH_SIZE];
>> @@ -221,6 +222,8 @@ static void init_beancounter_syslimits(s
>> {   int k;
>>
>> +   bc->bc_parms[BC_KMEMSIZE].limit = 32 * 1024 * 1024;
>> +
>
>
> Can't this be configurable CONFIG_XXX or a #defined constant?
This is some arbitraty limited container, just to make sure it is not
created unlimited. User space should initialize limits properly after creation
anyway. So I don't see reasons to make it configurable, do you?

>> --- ./mm/mempool.c.bckmem   2006-04-21 11:59:36.000000000 +0400
>> +++ ./mm/mempool.c   2006-08-28 12:59:28.000000000 +0400
>> @@ -119,6 +119,7 @@ int mempool_resize(mempool_t *pool, int
>> unsigned long flags;
>>
>>   BUG_ON(new_min_nr <= 0);
>> +   gfp_mask &= ~__GFP_BC;
>>
>>   spin_lock_irqsave(&pool->lock, flags);
>>   if (new_min_nr <= pool->min_nr) {
>> @@ -212,6 +213,7 @@ void * mempool_alloc(mempool_t *pool, gf
>>   gfp_mask |= __GFP_NOMEMALLOC;   /* don't allocate emergency
>> reserves */
>>   gfp_mask |= __GFP_NORETRY;   /* don't loop in __alloc_pages */
>>   gfp_mask |= __GFP_NOWARN;   /* failures are OK */
>> +   gfp_mask &= ~__GFP_BC;   /* do not charge */
>>
>>   gfp_temp = gfp_mask & ~(__GFP_WAIT|__GFP_IO);
>>
>
> Is there any reasn why mempool_xxxx() functions are not charged? Is it
> because
> mempool functions are mostly used from the I/O path?

yep.

>> --- ./mm/page_alloc.c.bckmem   2006-08-28 12:20:13.000000000 +0400
>> +++ ./mm/page_alloc.c   2006-08-28 12:59:28.000000000 +0400
>> @@ -40,6 +40,8 @@
>> #include <linux/sort.h>
>> #include <linux/pfn.h>
>>
>> +#include <bc/kmem.h>
>> +
>> #include <asm/tlbflush.h>
>> #include <asm/div64.h>
>> #include "internal.h"
>> @@ -516,6 +518,8 @@ static void __free_pages_ok(struct page   if
>> (reserved)
>>   return;
>>
>> +   bc_page_uncharge(page, order);
>> +
>>   kernel_map_pages(page, 1 << order, 0);
>>   local_irq_save(flags);
>>   __count_vm_events(PGFREE, 1 << order);
>> @@ -799,6 +803,8 @@ static void fastcall free_hot_cold_page(
>>   if (free_pages_check(page))
>>   return;
>>
>> +   bc_page_uncharge(page, 0);
>> +
>>   kernel_map_pages(page, 1, 0);
>>
>>   pcp = &zone_pcp(zone, get_cpu())->pcp[cold];
>> @@ -1188,6 +1194,11 @@ nopage:
>>   show_mem();
>>   }
>> got_pg:
>> +   if ((gfp_mask & __GFP_BC) &&
>> +   bc_page_charge(page, order, gfp_mask)) {
>
>
> I wonder if bc_page_charge() should be called bc_page_charge_failed()?
> Does it make sense to atleast partially start reclamation here? I know with
> bean counters we cannot reclaim from a particular container, but for now
> we could kick off kswapd() or call shrink_all_memory() inline (Dave's
> patches do this to shrink memory from the particular cpuset). Or do you
> want to leave this
> slot open for later?
yes. my intention is to account correctly all needed information first.
After we agree on accounting, we can agree on how to do reclamaition.

>> +		__free_pages(page, order);
>> +		page = NULL;
>> +	}
>
>
>

Subject: Re: [ckrm-tech] [PATCH 6/7] BC: kernel memory (core)
Posted by dev on Mon, 04 Sep 2006 12:20:00 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Tue, 2006-08-29 at 18:58 +0400, Kirill Korotaev wrote:
>
>>@@ -274,8 +274,14 @@ struct page {
>>		unsigned int gfp_mask;
>>		unsigned long trace[8];
>> #endif
>>+#ifdef CONFIG_BEANCOUNTERS
>>+	union {
>>+		struct beancounter	*page_bc;
>>+	} bc;
>>+#endif
>> };
>
>
> I know you're probably saving this union for when you put some userspace
> stuff in here or something. But, for now, it would probably be best
> just to leave it as a plain struct, or even an anonymous union.
>
> You probably had to use gcc 2 when this was written and couldn't use
> anonymous unions, right?
are you suggesting the code like this:
>>+#ifdef CONFIG_BEANCOUNTERS
>>+	union {
>>+		struct beancounter	*page_bc;
>>+	};
>>+#endif
>> };

#define page_bc(page) ((page)->page_bc)
?

> -- Dave
>
>

> ---------- ----------
> 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
>

Subject: Re: [PATCH 6/7] BC: kernel memory (core)
Posted by Balbir Singh on Mon, 04 Sep 2006 15:45:47 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
> Balbir Singh wrote:
>> Kirill Korotaev wrote:

>>> +#ifdef CONFIG_BEANCOUNTERS
>>> + union {
>>> + struct beancounter *page_bc;
>>> + } bc;
>>> +#endif
>>> };
>>>
>>> +#define page_bc(page) ((page)->bc.page_bc)
>>
>>
>> Minor comment - page->(bc).page_bc has too many repititions of page
>> and bc - see
>> the Practice of Programming by Kernighan and Pike
>>
>> I missed the part of why you wanted to have a union (in struct page
>> for bc)?
> because this union is used both for kernel memory accounting and user
> memeory tracking.

Ok.. that's good. I remember seeing a user_bc sometime back in the code.
I had some idea about allowing tasks to migrate across resources (bean
counters), which I think can be easily done for user space pages, if the
user limits are tracked separately.

>
>>> const char *bc_rnames[] = {
>>> + "kmemsize", /* 0 */
>>> };
>>>

>>> static struct hlist_head bc_hash[BC_HASH_SIZE];
>>> @@ -221,6 +222,8 @@ static void init_beancounter_syslimits(s
>>> {   int k;
>>>
>>> +   bc->bc_parms[BC_KMEMSIZE].limit = 32 * 1024 * 1024;
>>> +
>>
>>
>> Can't this be configurable CONFIG_XXX or a #defined constant?
> This is some arbitraty limited container, just to make sure it is not
> created unlimited. User space should initialize limits properly after
> creation
> anyway. So I don't see reasons to make it configurable, do you?

May be its not very important now but configurable limits will help a confused user. Even if we decide to use this number for now, a constant like BC_DEFAULT_MEM_LIMIT is easier to read.

>> I wonder if bc_page_charge() should be called bc_page_charge_failed()?
>> Does it make sense to atleast partially start reclamation here? I know
>> with
>> bean counters we cannot reclaim from a particular container, but for now
>> we could kick off kswapd() or call shrink_all_memory() inline (Dave's
>> patches do this to shrink memory from the particular cpuset). Or do
>> you want to leave this
>> slot open for later?
> yes. my intention is to account correctly all needed information first.
> After we agree on accounting, we can agree on how to do reclamaition.
>

That sounds like a good plan.

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH 6/7] BC: kernel memory (core)
Posted by Dave Hansen on Tue, 05 Sep 2006 16:21:34 GMT
View Forum Message <> Reply to Message

On Mon, 2006-09-04 at 16:23 +0400, Kirill Korotaev wrote:
>
> are you suggesting the code like this:
> >>+#ifdef CONFIG_BEANCOUNTERS
> >>+   union {

> >>+ struct beancounter *page_bc;
> >>+ };
> >>+#endif
> >> };
>
> #define page_bc(page) ((page)->page_bc)

That would be a bit better. Although, I think the "page_" bit is also superfluous.

-- Dave
