Subject: [PATCH] BC: resource beancounters (v5) (added userpages reclamation)
Posted by Kirill Korotaev on Thu, 05 Oct 2006 15:27:47 GMT
View Forum Message <> Reply to Message

MAJOR CHANGES:
- phys pages limit hit leads to user pages reclamation
- bc can be changed on arbitrary task

Core Resource Beancounters (BC) + kernel/user memory control.

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.

* diff-bc-syscalls.patch:
Patch adds system calls for BC management:
1. sys_get_bcid - get current BC id
2. sys_set_bcid - changes BC on task
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.

* diff-bc-vmpages-core:
Accounting of private pages (mappings), done per-vma.
Charged on mmap().

* diff-bc-rsspages-core:

Phys pages accounting. Charging is done on page touch
(page fault) and limit hit leads to reclamation.
Reclamation is done based on beancounter list of pages.

* diff-bc-userpages-charge:
Hooks across the kernel for user pages accounting.

* bcctl.c:
Tool for BC management. Allows changing BC, setting BC limits,
viewing current usages.

Summary of changes from v4 patch set:
* changed set of resources - kmemsize, privvmpages, physpages
* added event hooks for resources (init, limit hit etc)
* added user pages reclamation (bc_try_to_free_pages)
* removed pages sharing accounting - charge to first user
* task now carries only one BC pointer, simplified
* make set_bcid syscall move arbitrary task into BC
* resources are not recharged when task moves
* each vm_area_struct carries a BC pointer

Summary of changes from v3 patch set:

* Added basic user pages accounting (lockedpages/privvmpages)
* spell in Kconfig
* Makefile reworked
* EXPORT_SYMBOL_GPL
* union w/o name in struct page
* bc_task_charge is void now
* adjust minheld/maxheld splitted

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-mm3

Thanks,
Kirill

Subject: [PATCH 1/10] BC: introduce atomic_dec_and_lock_irqsave()
Posted by dev on Thu, 05 Oct 2006 15:46:42 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@openvz.org>
Signed-Off-By: Kirill Korotaev <dev@openvz.org>

```
 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 2/10] BC: kconfig
Posted by dev on Thu, 05 Oct 2006 15:46:55 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@openvz.org>
Signed-off-by: Kirill Korotaev <dev@openvz.org>

```
 init/Kconfig      |    4 ++++
 kernel/bc/Kconfig |   16 ++++++++++++++++
 2 files changed, 20 insertions(+)

--- ./init/Kconfig.bc_kconfig 2006-10-05 11:42:43.000000000 +0400
+++ ./init/Kconfig 2006-10-05 11:43:56.000000000 +0400
@@ -564,6 +564,10 @@ config STOP_MACHINE
    Need stop_machine() primitive.
 endmenu

+menu "Beancounters"
+source "kernel/bc/Kconfig"
+endmenu
+
 menu "Block layer"
 source "block/Kconfig"
 endmenu
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
```

+++ ./kernel/bc/Kconfig 2006-10-05 11:43:56.000000000 +0400
@@ -0,0 +1,16 @@
+config BEANCOUNTERS
+ bool "Enable resource accounting/control"
+ default n
+ help
+ When Y this option provides accounting and allows configuring
+ 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 accidental
+ misbehavior of processes or malicious activity aiming to ``kill''
+ the system. It's worth mentioning that resource limits configured
+ by setrlimit(2) do not give an acceptable level of protection
+ because they cover only a 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.

Subject: [PATCH 3/10] BC: beancounters core (API)
Posted by Kirill Korotaev on Thu, 05 Oct 2006 15:47:53 GMT
View Forum Message <> Reply to Message

Core functionality and interfaces of BCs:
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@openvz.org>
Signed-off-by: Kirill Korotaev <dev@openvz.org>

---

include/bc/beancounter.h | 169 ++
include/linux/types.h | 16 +++
init/main.c | 3
kernel/Makefile | 1
kernel/bc/Makefile | 12 ++
kernel/bc/beancounter.c | 214 ++
6 files changed, 415 insertions(+)

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/beancounter.h 2006-10-05 12:05:33.000000000 +0400
@@ -0,0 +1,169 @@

```
+/*
+ * include/bc/beancounter.h
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#ifndef __BEANCOUNTER_H__
+#define __BEANCOUNTER_H__
+
+#define BC_KMEMSIZE  0
+#define BC_PRIVVMPAGES  1
+#define BC_PHYSPAGES  2
+
+#define BC_RESOURCES 3
+
+struct bc_resource_parm {
+ unsigned long barrier;
+ unsigned long limit;
+ unsigned long held;
+ unsigned long minheld;
+ unsigned long maxheld;
+ unsigned long failcnt;
+};
+
+#ifdef __KERNEL__
+
+#include <linux/list.h>
+#include <linux/spinlock.h>
+#include <linux/init.h>
+#include <asm/atomic.h>
+
+#define BC_MAXVALUE ((unsigned long)LONG_MAX)
+
+enum bc_severity {
+ BC_BARRIER,
+ BC_LIMIT,
+ BC_FORCE,
+};
+
+#ifdef CONFIG_BEANCOUNTERS
+
+struct beancounter {
+ atomic_t  bc_refcount;
+ spinlock_t  bc_lock;
+ bcid_t   bc_id;
+ struct hlist_node bc_hash;
+
```

```
+ struct bc_resource_parm bc_parms[BC_RESOURCES];
+
+ struct list_head bc_page_list;
+ spinlock_t bc_page_lock;
+};
+
+struct bc_resource {
+ char *bcr_name;
+
+ int (*bcr_init)(struct beancounter *bc, gfp_t mask);
+ int (*bcr_change)(struct beancounter *bc,
+   unsigned long new_bar, unsigned long new_lim);
+ void (*bcr_barrier_hit)(struct beancounter *bc);
+ int (*bcr_limit_hit)(struct beancounter *bc, unsigned long val,
+   unsigned long flags);
+ void (*bcr_fini)(struct beancounter *bc);
+};
+
+extern struct bc_resource *bc_resources[BC_RESOURCES];
+
+static inline struct beancounter *bc_get(struct beancounter *bc)
+{
+ atomic_inc(&bc->bc_refcount);
+ return bc;
+}
+extern void bc_put(struct beancounter *bc);
+
+#define BC_LOOKUP  0 /* Just lookup in hash
+       */
+#define BC_ALLOC  1 /* Lookup in hash and try to make
+       * new BC if no one found
+       */
+#define BC_ALLOC_ATOMIC  2 /* When BC_ALLOC is set perform
+       * GFP_ATOMIC allocation
+       */
+
+extern struct beancounter *bc_findcreate(bcid_t bcid, int bc_flags);
+
+static inline void bc_adjust_maxheld(struct bc_resource_parm *parm)
+{
+ if (parm->maxheld < parm->held)
+  parm->maxheld = parm->held;
+}
+
+static inline void bc_adjust_minheld(struct bc_resource_parm *parm)
+{
+ if (parm->minheld > parm->held)
+  parm->minheld = parm->held;
```

```
+}
+
+static inline void bc_init_resource(struct beancounter *bc, int res_id,
+  unsigned long bar, unsigned long lim)
+{
+ struct bc_resource_parm *parm;
+
+ parm = &bc->bc_parms[res_id];
+
+ parm->barrier = bar;
+ parm->limit = lim;
+ parm->held = 0;
+ parm->minheld = 0;
+ parm->maxheld = 0;
+ parm->failcnt = 0;
+}
+
+int __must_check bc_charge_locked(struct beancounter *bc, int res_id,
+  unsigned long val, int strict, unsigned long flags);
+static inline int __must_check bc_charge(struct beancounter *bc, int res_id,
+  unsigned long val, int strict)
+{
+ unsigned long flags;
+ int ret;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ ret = bc_charge_locked(bc, res_id, val, strict, flags);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+ return ret;
+}
+
+void bc_uncharge_locked(struct beancounter *bc, int res_id, unsigned long val);
+static inline void bc_uncharge(struct beancounter *bc, int res_id,
+  unsigned long val)
+{
+ unsigned long flags;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ bc_uncharge_locked(bc, res_id, val);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+}
+
+void __init bc_register_resource(int res_id, struct bc_resource *br);
+void __init bc_init_early(void);
+#else /* CONFIG_BEANCOUNTERS */
+static inline int __must_check bc_charge_locked(struct beancounter *bc, int res,
+  unsigned long val, int strict, unsigned long flags)
+{
```

```
+ return 0;
+}
+
+static inline int __must_check bc_charge(struct beancounter *bc, int res,
+  unsigned long val, int strict)
+{
+ return 0;
+}
+
+static inline void bc_uncharge_locked(struct beancounter *bc, int res,
+  unsigned long val)
+{
+}
+
+static inline void bc_uncharge(struct beancounter *bc, int res,
+  unsigned long val)
+{
+}
+
+static inline void bc_init_early(void)
+{
+}
+#endif /* CONFIG_BEANCOUNTERS */
+#endif /* __KERNEL__ */
+#endif
--- ./include/linux/types.h.bc_core 2006-10-05 11:42:43.000000000 +0400
+++ ./include/linux/types.h 2006-10-05 11:44:32.000000000 +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.bc_core 2006-10-05 11:42:43.000000000 +0400
+++ ./init/main.c 2006-10-05 11:44:32.000000000 +0400
@@ -50,6 +50,8 @@
 #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>
@@ -480,6 +482,7 @@ asmlinkage void __init start_kernel(void
  char * command_line;
  extern struct kernel_param __start___param[], __stop___param[];

+ bc_init_early();
  smp_setup_processor_id();

  /*
--- ./kernel/Makefile.bc_core 2006-10-05 11:42:43.000000000 +0400
+++ ./kernel/Makefile 2006-10-05 11:44:32.000000000 +0400
@@ -12,6 +12,7 @@ obj-y     = sched.o fork.o exec_domain.o

 obj-$(CONFIG_STACKTRACE) += stacktrace.o
 obj-y += time/
+obj-$(CONFIG_BEANCOUNTERS) += 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 2006-10-05 11:44:32.000000000 +0400
@@ -0,0 +1,12 @@
+#
+# kernel/bc/Makefile
+#
+# Copyright (C) 2006 OpenVZ SWsoft Inc.
+#
+obj-y = beancounter.o
+
```

```
+obj-y += sys.o
+obj-y += misc.o
+obj-y += kmem.o
+obj-y += vmpages.o
+obj-y += rsspages.o
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/beancounter.c 2006-10-05 12:02:19.000000000 +0400
@@ -0,0 +1,214 @@
+/*
+ * kernel/bc/beancounter.c
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#include <linux/sched.h>
+#include <linux/list.h>
+#include <linux/hash.h>
+#include <linux/gfp.h>
+#include <linux/slab.h>
+
+#include <bc/beancounter.h>
+#include <bc/task.h>
+
+#define BC_HASH_BITS (8)
+#define BC_HASH_SIZE (1 << BC_HASH_BITS)
+
+struct bc_resource *bc_resources[BC_RESOURCES];
+
+struct beancounter init_bc;
+
+static struct hlist_head bc_hash[BC_HASH_SIZE];
+static spinlock_t bc_hash_lock;
+static kmem_cache_t *bc_cache;
+
+static void init_beancounter_struct(struct beancounter *bc, bcid_t bcid)
+{
+ bc->bc_id = bcid;
+ spin_lock_init(&bc->bc_lock);
+ atomic_set(&bc->bc_refcount, 1);
+}
+
+struct beancounter *bc_findcreate(bcid_t bcid, int bc_flags)
+{
+ unsigned long flags;
+ struct beancounter *bc;
+ struct beancounter *new_bc;
+ struct hlist_head *head;
```

```
+ struct hlist_node *ptr;
+ gfp_t mask;
+ int i;
+
+ head = &bc_hash[hash_long(bcid, BC_HASH_BITS)];
+ bc = NULL;
+ new_bc = NULL;
+ mask = ((bc_flags & BC_ALLOC_ATOMIC) ? GFP_ATOMIC : GFP_KERNEL);
+
+retry:
+ spin_lock_irqsave(&bc_hash_lock, flags);
+ hlist_for_each (ptr, head) {
+  bc = hlist_entry(ptr, struct beancounter, bc_hash);
+  if (bc->bc_id == bcid)
+   break;
+ }
+
+ if (bc != NULL) {
+  bc_get(bc);
+  spin_unlock_irqrestore(&bc_hash_lock, flags);
+
+  if (new_bc != NULL)
+   kmem_cache_free(bc_cache, new_bc);
+  return bc;
+ }
+
+ if (new_bc != NULL) {
+  hlist_add_head(&new_bc->bc_hash, head);
+  spin_unlock_irqrestore(&bc_hash_lock, flags);
+  return new_bc;
+ }
+ spin_unlock_irqrestore(&bc_hash_lock, flags);
+
+ if (!(bc_flags & BC_ALLOC))
+  return NULL;
+
+ new_bc = kmem_cache_alloc(bc_cache, mask);
+ if (new_bc == NULL)
+  return NULL;
+
+ init_beancounter_struct(new_bc, bcid);
+ for (i = 0; i < BC_RESOURCES; i++)
+  if (bc_resources[i]->bcr_init(new_bc, mask))
+   goto out_unroll;
+ goto retry;
+
+out_unroll:
+ for (i--; i >= 0; i--)
```

```
+  if (bc_resources[i]->bcr_fini)
+   bc_resources[i]->bcr_fini(new_bc);
+ kmem_cache_free(bc_cache, new_bc);
+ return NULL;
+}
+
+void bc_put(struct beancounter *bc)
+{
+ int i;
+ unsigned long flags;
+
+ if (likely(!atomic_dec_and_lock_irqsave(&bc->bc_refcount,
+    &bc_hash_lock, flags)))
+  return;
+
+ hlist_del(&bc->bc_hash);
+ spin_unlock_irqrestore(&bc_hash_lock, flags);
+
+ for (i = 0; i < BC_RESOURCES; i++) {
+  if (bc_resources[i]->bcr_fini)
+   bc_resources[i]->bcr_fini(bc);
+  if (bc->bc_parms[i].held != 0)
+   printk(KERN_ERR "BC: Resource %s holds %lu on put\n",
+     bc_resources[i]->bcr_name,
+     bc->bc_parms[i].held);
+ }
+
+ kmem_cache_free(bc_cache, bc);
+}
+
+int bc_charge_locked(struct beancounter *bc, int res, unsigned long val,
+  int strict, unsigned long flags)
+{
+ struct bc_resource_parm *parm;
+ unsigned long new_held;
+
+ BUG_ON(val > BC_MAXVALUE);
+
+ parm = &bc->bc_parms[res];
+ new_held = parm->held + val;
+
+ switch (strict) {
+ case BC_LIMIT:
+  if (new_held > parm->limit)
+   break;
+  /* fallthrough */
+ case BC_BARRIER:
+  if (new_held > parm->barrier) {
```

```
+   if (strict == BC_BARRIER)
+    break;
+   if (parm->held < parm->barrier &&
+     bc_resources[res]->bcr_barrier_hit)
+    bc_resources[res]->bcr_barrier_hit(bc);
+  }
+  /* fallthrough */
+ case BC_FORCE:
+  parm->held = new_held;
+  bc_adjust_maxheld(parm);
+  return 0;
+ default:
+  BUG();
+ }
+
+ if (bc_resources[res]->bcr_limit_hit)
+  return bc_resources[res]->bcr_limit_hit(bc, val, flags);
+
+ parm->failcnt++;
+ return -ENOMEM;
+}
+
+void bc_uncharge_locked(struct beancounter *bc, int res, unsigned long val)
+{
+ struct bc_resource_parm *parm;
+
+ BUG_ON(val > BC_MAXVALUE);
+
+ parm = &bc->bc_parms[res];
+ if (unlikely(val > parm->held)) {
+  printk(KERN_ERR "BC: Uncharging too much of %s: %lu vs %lu\n",
+    bc_resources[res]->bcr_name,
+    val, parm->held);
+  val = parm->held;
+ }
+
+ parm->held -= val;
+ bc_adjust_minheld(parm);
+}
+
+void __init bc_register_resource(int res_id, struct bc_resource *br)
+{
+ BUG_ON(bc_resources[res_id] != NULL);
+ BUG_ON(res_id >= BC_RESOURCES);
+
+ bc_resources[res_id] = br;
+ printk(KERN_INFO "BC: Registered %s bc resource\n", br->bcr_name);
+}
```

```
+
+void __init bc_init_early(void)
+{
+ int i;
+
+ init_beancounter_struct(&init_bc, 0);
+ spin_lock_init(&init_bc.bc_page_lock);
+ INIT_LIST_HEAD(&init_bc.bc_page_list);
+ for (i = 0; i < BC_RESOURCES; i++) {
+  init_bc.bc_parms[i].barrier = BC_MAXVALUE;
+  init_bc.bc_parms[i].limit = BC_MAXVALUE;
+ }
+
+ spin_lock_init(&bc_hash_lock);
+ hlist_add_head(&init_bc.bc_hash, &bc_hash[hash_long(0, BC_HASH_BITS)]);
+
+ current->exec_bc = bc_get(&init_bc);
+ init_mm.mm_bc = bc_get(&init_bc);
+}
+
+int __init bc_init_late(void)
+{
+ bc_cache = kmem_cache_create("beancounters",
+    sizeof(struct beancounter), 0,
+    SLAB_HWCACHE_ALIGN | SLAB_PANIC, NULL, NULL);
+ printk(KERN_INFO "BC: Kmem cache created\n");
+ return 0;
+}
+
+__initcall(bc_init_late);
```

Subject: [PATCH 4/10] BC: context inheriting and changing
Posted by Kirill Korotaev on Thu, 05 Oct 2006 15:50:05 GMT
View Forum Message <> Reply to Message

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

Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Signed-off-by: Kirill Korotaev <dev@openvz.org>

```
 include/bc/task.h     |   51 ++++++++++++++++++++++++++++++++++++++++++++++++
 include/linux/sched.h |    5 +++
 kernel/bc/misc.c      |   70 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 kernel/fork.c         |    5 +++
```

```
 kernel/irq/handle.c   |    6 ++++
 kernel/softirq.c      |    5 +++
 6 files changed, 142 insertions(+)

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/task.h 2006-10-05 12:11:44.000000000 +0400
@@ -0,0 +1,51 @@
+/*
+ * include/bc/beancounter.h
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#ifndef __BC_TASK_H__
+#define __BC_TASK_H__
+
+struct beancounter;
+struct task_struct;
+
+#ifdef CONFIG_BEANCOUNTERS
+#define get_exec_bc() (current->exec_bc)
+
+#define set_exec_bc(bc) ({   \
+  struct task_struct *t;  \
+  struct beancounter *old; \
+  t = current;   \
+  old = t->exec_bc;  \
+  t->exec_bc = bc;  \
+  old;    \
+ })
+
+#define reset_exec_bc(old, expected) do { \
+  struct task_struct *t;  \
+  t = current;   \
+  BUG_ON(t->exec_bc != expected);\
+  t->exec_bc = old;  \
+ } while (0)
+
+extern struct beancounter init_bc;
+
+void copy_beancounter(struct task_struct *tsk, struct task_struct *parent);
+void free_beancounter(struct task_struct *tsk);
+int bc_task_move(struct task_struct *tsk, struct beancounter *bc);
+#else
+static inline void copy_beancounter(struct task_struct *tsk,
+  struct task_struct *parent)
+{
```

```
+}
+
+static inline void free_beancounter(struct task_struct *tsk)
+{
+}
+
+#define set_exec_bc(bc)  (NULL)
+#define reset_exec_bc(bc, exp) do { } while (0)
+#endif
+#endif
--- ./include/linux/sched.h.bc_task 2006-10-05 11:42:43.000000000 +0400
+++ ./include/linux/sched.h 2006-10-05 12:11:44.000000000 +0400
@@ -77,6 +77,8 @@ struct sched_param {
 #include <linux/futex.h>
 #include <linux/rtmutex.h>

+#include <bc/task.h>
+
 #include <linux/time.h>
 #include <linux/param.h>
 #include <linux/resource.h>
@@ -1055,6 +1057,9 @@ struct task_struct {
 #ifdef CONFIG_TASK_DELAY_ACCT
  struct task_delay_info *delays;
 #endif
+#ifdef CONFIG_BEANCOUNTERS
+ struct beancounter *exec_bc;
+#endif
 };

 static inline pid_t process_group(struct task_struct *tsk)
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/misc.c 2006-10-05 12:11:44.000000000 +0400
@@ -0,0 +1,70 @@
+/*
+ * kernel/bc/misc.c
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#include <linux/sched.h>
+#include <linux/stop_machine.h>
+
+#include <bc/beancounter.h>
+#include <bc/task.h>
+
+void copy_beancounter(struct task_struct *tsk, struct task_struct *parent)
```

```
+{
+ tsk->exec_bc = bc_get(parent->exec_bc);
+}
+
+void free_beancounter(struct task_struct *tsk)
+{
+ bc_put(tsk->exec_bc);
+ tsk->exec_bc = NULL;
+}
+
+struct set_bcid_data {
+ struct beancounter *bc;
+ struct mm_struct *mm;
+};
+
+static int do_set_bcid(void *data)
+{
+ struct set_bcid_data *d;
+ struct mm_struct *mm;
+ struct task_struct *g, *p;
+
+ d = (struct set_bcid_data *)data;
+ mm = d->mm;
+
+ do_each_thread (g, p) {
+  if (p->mm == mm) {
+   bc_put(p->exec_bc);
+   p->exec_bc = bc_get(d->bc);
+  }
+ } while_each_thread (g, p);
+
+ bc_put(mm->mm_bc);
+ mm->mm_bc = bc_get(d->bc);
+ return 0;
+}
+
+int bc_task_move(struct task_struct *tsk, struct beancounter *bc)
+{
+ int err;
+ struct set_bcid_data data;
+ struct mm_struct *mm;
+
+ mm = get_task_mm(tsk);
+ if (mm == NULL)
+  return -EINVAL;
+
+ data.bc = bc;
+ data.mm = mm;
```

```
+
+ down_write(&mm->mmap_sem);
+ err = stop_machine_run(do_set_bcid, &data, NR_CPUS);
+ up_write(&mm->mmap_sem);
+
+ mmput(mm);
+ return err;
+}
--- ./kernel/fork.c.bc_task 2006-10-05 11:42:43.000000000 +0400
+++ ./kernel/fork.c 2006-10-05 12:11:44.000000000 +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>
@@ -105,6 +107,7 @@ static kmem_cache_t *mm_cachep;

 void free_task(struct task_struct *tsk)
 {
+ free_beancounter(tsk);
  free_thread_info(tsk->thread_info);
  rt_mutex_debug_task_free(tsk);
  free_task_struct(tsk);
@@ -984,6 +987,8 @@ static struct task_struct *copy_process(
  if (!p)
   goto fork_out;

+ copy_beancounter(p, current);
+
 #ifdef CONFIG_TRACE_IRQFLAGS
  DEBUG_LOCKS_WARN_ON(!p->hardirqs_enabled);
  DEBUG_LOCKS_WARN_ON(!p->softirqs_enabled);
--- ./kernel/irq/handle.c.bc_task 2006-10-05 11:42:43.000000000 +0400
+++ ./kernel/irq/handle.c 2006-10-05 12:11:44.000000000 +0400
@@ -16,6 +16,8 @@
 #include <linux/interrupt.h>
 #include <linux/kernel_stat.h>

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

 /**
@@ -172,6 +174,7 @@ fastcall unsigned int __do_IRQ(unsigned
```

```
  struct irq_desc *desc = irq_desc + irq;
  struct irqaction *action;
  unsigned int status;
+ struct beancounter *bc;

  kstat_this_cpu.irqs[irq]++;
  if (CHECK_IRQ_PER_CPU(desc->status)) {
@@ -228,6 +231,8 @@ fastcall unsigned int __do_IRQ(unsigned
   * useful for irq hardware that does not mask cleanly in an
   * SMP environment.
   */
+
+ bc = set_exec_bc(&init_bc);
  for (;;) {
   irqreturn_t action_ret;

@@ -242,6 +247,7 @@ fastcall unsigned int __do_IRQ(unsigned
    break;
   desc->status &= ~IRQ_PENDING;
  }
+ reset_exec_bc(bc, &init_bc);
  desc->status &= ~IRQ_INPROGRESS;

 out:
--- ./kernel/softirq.c.bc_task 2006-10-05 11:42:43.000000000 +0400
+++ ./kernel/softirq.c 2006-10-05 12:11:44.000000000 +0400
@@ -18,6 +18,8 @@
 #include <linux/rcupdate.h>
 #include <linux/smp.h>

+#include <bc/task.h>
+
 #include <asm/irq.h>
 /*
    - No shared variables, all the data are CPU local.
@@ -209,6 +211,7 @@ asmlinkage void __do_softirq(void)
  __u32 pending;
  int max_restart = MAX_SOFTIRQ_RESTART;
  int cpu;
+ struct beancounter *bc;

  pending = local_softirq_pending();
  account_system_vtime(current);
@@ -225,6 +228,7 @@ restart:

  h = softirq_vec;

+ bc = set_exec_bc(&init_bc);
```

```
  do {
   if (pending & 1) {
    h->action(h);
@@ -233,6 +237,7 @@ restart:
   h++;
   pending >>= 1;
  } while (pending);
+ reset_exec_bc(bc, &init_bc);

  local_irq_disable();
```

Subject: [PATCH 5/10] BC: user interface (syscalls)
Posted by Kirill Korotaev on Thu, 05 Oct 2006 15:50:57 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 BC on task
 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@openvz.org>
Signed-off-by: Kirill Korotaev <dev@openvz.org>

```
 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/sys.c                  |  130 ++++++++++++++++++++++++++++++++++++++++
 kernel/sys_ni.c                  |    6 +
 15 files changed, 195 insertions(+), 9 deletions(-)

--- ./arch/i386/kernel/syscall_table.S.bc_syscalls 2006-10-05 11:42:39.000000000 +0400
+++ ./arch/i386/kernel/syscall_table.S 2006-10-05 12:09:09.000000000 +0400
@@ -322,3 +322,7 @@ ENTRY(sys_call_table)
  .long sys_kevent_get_events /* 320 */
```

```
  .long sys_kevent_ctl
  .long sys_kevent_wait
+ .long sys_get_bcid
+ .long sys_set_bcid
+ .long sys_set_bclimit  /* 325 */
+ .long sys_get_bcstat
--- ./arch/ia64/kernel/entry.S.bc_syscalls 2006-10-05 11:42:39.000000000 +0400
+++ ./arch/ia64/kernel/entry.S 2006-10-05 12:09:09.000000000 +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.bc_syscalls 2006-09-20 14:46:17.000000000 +0400
+++ ./arch/sparc/kernel/entry.S 2006-10-05 12:09:09.000000000 +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.bc_syscalls 2006-09-20 14:46:17.000000000 +0400
+++ ./arch/sparc/kernel/systbls.S 2006-10-05 12:09:09.000000000 +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.bc_syscalls 2006-09-20 14:46:17.000000000 +0400
+++ ./arch/sparc64/kernel/entry.S 2006-10-05 12:09:09.000000000 +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.bc_syscalls 2006-09-20 14:46:17.000000000 +0400
+++ ./arch/sparc64/kernel/systbls.S 2006-10-05 12:09:09.000000000 +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, 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.bc_syscalls 2006-10-05 11:42:42.000000000 +0400
+++ ./include/asm-i386/unistd.h 2006-10-05 12:09:09.000000000 +0400
```

```
@@ -328,10 +328,14 @@
 #define __NR_kevent_get_events 320
 #define __NR_kevent_ctl  321
 #define __NR_kevent_wait 322
+#define __NR_get_bcid  323
+#define __NR_set_bcid  324
+#define __NR_set_bclimit 325
+#define __NR_get_bcstat  326

 #ifdef __KERNEL__

-#define NR_syscalls 323
+#define NR_syscalls 327
 #include <linux/err.h>

 /*
--- ./include/asm-ia64/unistd.h.bc_syscalls 2006-10-05 11:42:42.000000000 +0400
+++ ./include/asm-ia64/unistd.h 2006-10-05 12:09:09.000000000 +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.bc_syscalls 2006-09-20 14:46:39.000000000 +0400
+++ ./include/asm-powerpc/systbl.h 2006-10-05 12:09:09.000000000 +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.bc_syscalls 2006-10-05 11:42:42.000000000 +0400
+++ ./include/asm-powerpc/unistd.h 2006-10-05 12:09:09.000000000 +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.bc_syscalls 2006-10-05 11:42:43.000000000 +0400
+++ ./include/asm-sparc/unistd.h 2006-10-05 12:09:09.000000000 +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.bc_syscalls 2006-10-05 11:42:43.000000000 +0400
+++ ./include/asm-sparc64/unistd.h 2006-10-05 12:09:09.000000000 +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.bc_syscalls 2006-10-05 11:42:43.000000000 +0400
+++ ./include/asm-x86_64/unistd.h 2006-10-05 12:09:09.000000000 +0400
@@ -625,8 +625,16 @@ __SYSCALL(__NR_kevent_get_events, sys_ke
 __SYSCALL(__NR_kevent_ctl, sys_kevent_ctl)
 #define __NR_kevent_wait 282
 __SYSCALL(__NR_kevent_wait, sys_kevent_wait)
+#define __NR_get_bcid  283
```

```
+__SYSCALL(__NR_get_bcid, sys_get_bcid)
+#define __NR_set_bcid  284
+__SYSCALL(__NR_set_bcid, sys_set_bcid)
+#define __NR_set_bclimit 285
+__SYSCALL(__NR_set_bclimit, sys_set_bclimit)
+#define __NR_get_bcstat  286
+__SYSCALL(__NR_get_bcstat, sys_get_bcstat)

-#define __NR_syscall_max __NR_kevent_wait
+#define __NR_syscall_max __NR_get_bcstat

 #ifdef __KERNEL__
 #include <linux/err.h>
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/sys.c 2006-10-05 12:10:31.000000000 +0400
@@ -0,0 +1,130 @@
+/*
+ * 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, pid_t pid)
+{
+ int error;
+ struct task_struct *tsk;
+ struct beancounter *bc;
+
+ error = -EPERM;
+ if (!capable(CAP_SETUID))
+  goto out;
+
```

```
+ error = -ENOMEM;
+ bc = bc_findcreate(id, BC_ALLOC);
+ if (bc == NULL)
+  goto out;
+
+ read_lock(&tasklist_lock);
+ tsk = find_task_by_pid(pid);
+ if (tsk != NULL)
+  get_task_struct(tsk);
+ read_unlock(&tasklist_lock);
+
+ error = -ESRCH;
+ if (tsk == NULL)
+  goto out_putbc;
+
+ error = bc_task_move(tsk, bc);
+
+ put_task_struct(tsk);
+out_putbc:
+ bc_put(bc);
+out:
+ return error;
+}
+
+asmlinkage long sys_set_bclimit(bcid_t id, unsigned long resource,
+  unsigned long __user *limits)
+{
+ int error;
+ 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 = bc_findcreate(id, BC_LOOKUP);
+ if (bc == NULL)
+  goto out;
+
+ spin_lock_irq(&bc->bc_lock);
+ error = 0;
+ if (bc_resources[resource]->bcr_change)
+  error = bc_resources[resource]->bcr_change(bc,
+    new_limits[0], new_limits[1]);
+ if (error < 0)
+  goto out_unlock;
+
+ bc->bc_parms[resource].barrier = new_limits[0];
+ bc->bc_parms[resource].limit = new_limits[1];
+ spin_unlock_irq(&bc->bc_lock);
+out_unlock:
+ bc_put(bc);
+out:
+ return error;
+}
+
+asmlinkage long sys_get_bcstat(bcid_t id, unsigned long resource,
+  struct bc_resource_parm __user *uparm)
+{
+ int error;
+ struct beancounter *bc;
+ struct bc_resource_parm parm;
+
+ error = -EINVAL;
+ if (resource >= BC_RESOURCES)
+  goto out;
+
+ error = -ENOENT;
+ bc = bc_findcreate(id, BC_LOOKUP);
+ if (bc == NULL)
+  goto out;
+
+ spin_lock_irq(&bc->bc_lock);
+ parm = bc->bc_parms[resource];
+ spin_unlock_irq(&bc->bc_lock);
+ bc_put(bc);
+
+ error = 0;
+ if (copy_to_user(uparm, &parm, sizeof(parm)))
+  error = -EFAULT;
+
+out:
```

```
+ return error;
+}
--- ./kernel/sys_ni.c.bc_syscalls 2006-10-05 11:42:43.000000000 +0400
+++ ./kernel/sys_ni.c 2006-10-05 12:09:09.000000000 +0400
@@ -143,3 +143,9 @@ cond_syscall(compat_sys_move_pages);
 cond_syscall(sys_bdflush);
 cond_syscall(sys_ioprio_set);
 cond_syscall(sys_ioprio_get);
+
+/* user resources syscalls */
+cond_syscall(sys_set_bcid);
+cond_syscall(sys_get_bcid);
+cond_syscall(sys_set_bclimit);
+cond_syscall(sys_get_bcstat);
```

Subject: [PATCH 6/10] BC: kernel memory (core)
Posted by Kirill Korotaev on Thu, 05 Oct 2006 15:52:13 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@openvz.org>
Signed-off-by: Kirill Korotaev <dev@openvz.org>

```
 include/bc/kmem.h        |   48 ++++++++++++++++++++
 include/linux/gfp.h      |    8 ++-
 include/linux/mm.h       |    1
 include/linux/mm_types.h |    3 +
 include/linux/slab.h     |    4 +
 include/linux/vmalloc.h  |    1
 kernel/bc/kmem.c         |  112 +++++++++++++++++++++++++++++++++++++++++++++++
 mm/mempool.c             |    2
```

```
 mm/page_alloc.c         |   11 ++++
 mm/slab.c               |  112 ++++++++++++++++++++++++++++++++++++++++--------
 mm/vmalloc.c            |    6 ++
 11 files changed, 287 insertions(+), 21 deletions(-)

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/kmem.h 2006-10-05 12:13:58.000000000 +0400
@@ -0,0 +1,48 @@
+/*
+ * include/bc/kmem.h
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#ifndef __BC_KMEM_H_
+#define __BC_KMEM_H_
+
+/*
+ * BC_KMEMSIZE accounting
+ */
+
+#include <linux/slab.h>
+#include <linux/gfp.h>
+
+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.bc_kmem_core 2006-10-05 11:42:43.000000000 +0400
+++ ./include/linux/gfp.h 2006-10-05 12:12:24.000000000 +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.bc_kmem_core 2006-10-05 11:42:43.000000000 +0400
+++ ./include/linux/mm.h 2006-10-05 12:12:24.000000000 +0400
@@ -219,6 +219,7 @@ struct vm_operations_struct {
 struct mmu_gather;
 struct inode;

+#define page_bc(page)   ((page)->bc)
 #define page_private(page)  ((page)->private)
```

```
 #define set_page_private(page, v) ((page)->private = (v))

--- ./include/linux/mm_types.h.bc_kmem_core 2006-10-05 11:42:43.000000000 +0400
+++ ./include/linux/mm_types.h 2006-10-05 12:12:24.000000000 +0400
@@ -62,6 +62,9 @@ struct page {
  void *virtual;   /* Kernel virtual address (NULL if
        not kmapped, ie. highmem) */
 #endif /* WANT_PAGE_VIRTUAL */
+#ifdef CONFIG_BEANCOUNTERS
+ struct beancounter *bc;
+#endif
 #ifdef CONFIG_PAGE_OWNER
  int order;
  unsigned int gfp_mask;
--- ./include/linux/slab.h.bc_kmem_core 2006-10-05 11:42:43.000000000 +0400
+++ ./include/linux/slab.h 2006-10-05 12:12:24.000000000 +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 */
@@ -305,6 +307,8 @@ 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.bc_kmem_core 2006-10-05 11:42:43.000000000 +0400
+++ ./include/linux/vmalloc.h 2006-10-05 12:12:24.000000000 +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);
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/kmem.c 2006-10-05 12:13:35.000000000 +0400
@@ -0,0 +1,112 @@
+/*
```

```
+ * 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>
+
+#define BC_KMEMSIZE_BARRIER (64 * 1024)
+#define BC_KMEMSIZE_LIMIT (64 * 1024)
+
+/*
+ * 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);
+ BUG_ON(*slab_bcp != NULL);
+ *slab_bcp = bc_get(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);
+ bc_put(bc);
+ *slab_bcp = NULL;
+}
+
+/*
+ * Pages accounting
+ */
+
+int bc_page_charge(struct page *page, int order, gfp_t flags)
+{
+ struct beancounter *bc;
+
+ bc = get_exec_bc();
+
+ if (bc_charge(bc, BC_KMEMSIZE, PAGE_SIZE << order,
+    (flags & __GFP_BC_LIMIT ? BC_LIMIT : BC_BARRIER)))
+  return -ENOMEM;
+
+ BUG_ON(page_bc(page) != NULL);
+ page_bc(page) = bc_get(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);
+ bc_put(bc);
+ page_bc(page) = NULL;
+}
+
+/*
+ * Generic resource info
+ */
+
+static int bc_kmem_init(struct beancounter *bc, gfp_t mask)
+{
+ bc_init_resource(bc, BC_KMEMSIZE,
+    BC_KMEMSIZE_BARRIER, BC_KMEMSIZE_LIMIT);
```

```
+ return 0;
+}
+
+struct bc_resource bc_kmem_resource = {
+ .bcr_name  = "kmemsize",
+ .bcr_init  = bc_kmem_init,
+};
+
+static int __init bc_kmem_init_resource(void)
+{
+ bc_register_resource(BC_KMEMSIZE, &bc_kmem_resource);
+ return 0;
+}
+
+__initcall(bc_kmem_init_resource);
--- ./mm/mempool.c.bc_kmem_core 2006-09-20 14:46:41.000000000 +0400
+++ ./mm/mempool.c 2006-10-05 12:12:24.000000000 +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.bc_kmem_core 2006-10-05 11:42:43.000000000 +0400
+++ ./mm/page_alloc.c 2006-10-05 12:12:24.000000000 +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"
@@ -517,6 +519,8 @@ 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);
@@ -800,6 +804,8 @@ 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];
@@ -1190,6 +1196,11 @@ 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.bc_kmem_core 2006-10-05 11:42:43.000000000 +0400
+++ ./mm/slab.c 2006-10-05 12:12:24.000000000 +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;
@@ -1965,7 +1994,8 @@ 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;
@@ -2273,8 +2303,8 @@ 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
@@ -2285,11 +2315,9 @@ 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. */
@@ -2535,6 +2563,30 @@ 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
@@ -2555,7 +2607,8 @@ 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 {
@@ -2566,14 +2619,14 @@ 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)
 {
@@ -2751,7 +2804,7 @@ 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;

@@ -3094,6 +3147,19 @@ 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)
 {
@@ -3119,6 +3185,12 @@ 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;
 }
@@ -3342,6 +3414,8 @@ 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.bc_kmem_core 2006-10-05 11:42:43.000000000 +0400
+++ ./mm/vmalloc.c 2006-10-05 12:12:24.000000000 +0400
@@ -512,6 +512,12 @@ 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 zeroed virtually contiguous memory for userspace
  * @size: allocation size
```

Subject: [PATCH 7/10] BC: kernel memory (marks)
Posted by dev on Thu, 05 Oct 2006 15:53:10 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@openvz.org>
Signed-off-by: Kirill Korotaev <dev@openvz.org>

```
 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 ++-
 mm/slab.c                       |    9 ++++++---
 22 files changed, 86 insertions(+), 59 deletions(-)

--- ./arch/i386/kernel/ldt.c.bc_kmem_charge 2006-10-05 11:42:39.000000000 +0400
+++ ./arch/i386/kernel/ldt.c 2006-10-05 12:14:35.000000000 +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.bc_kmem_charge 2006-10-05 11:42:39.000000000 +0400
+++ ./arch/i386/mm/init.c 2006-10-05 12:14:35.000000000 +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.bc_kmem_charge 2006-10-05 11:42:39.000000000 +0400
+++ ./arch/i386/mm/pgtable.c 2006-10-05 12:14:35.000000000 +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.bc_kmem_charge 2006-10-05 11:42:40.000000000 +0400
+++ ./drivers/char/tty_io.c 2006-10-05 12:14:35.000000000 +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.bc_kmem_charge 2006-10-05 11:42:42.000000000 +0400
+++ ./fs/file.c 2006-10-05 12:14:35.000000000 +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.bc_kmem_charge 2006-10-05 11:42:42.000000000 +0400
+++ ./fs/locks.c 2006-10-05 12:14:35.000000000 +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.bc_kmem_charge 2006-10-05 11:42:42.000000000 +0400
+++ ./fs/namespace.c 2006-10-05 12:14:35.000000000 +0400
@@ -1812,7 +1812,8 @@ 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.bc_kmem_charge 2006-10-05 11:42:42.000000000 +0400
+++ ./fs/select.c 2006-10-05 12:14:35.000000000 +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;
  }
```

```
@@ -687,7 +688,7 @@ 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.bc_kmem_charge 2006-09-20 14:46:38.000000000 +0400
+++ ./include/asm-i386/thread_info.h 2006-10-05 12:14:35.000000000 +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.bc_kmem_charge 2006-09-20 14:46:38.000000000 +0400
+++ ./include/asm-ia64/pgalloc.h 2006-10-05 12:14:35.000000000 +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.bc_kmem_charge 2006-09-20 14:46:40.000000000 +0400
+++ ./include/asm-x86_64/pgalloc.h 2006-10-05 12:14:35.000000000 +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.bc_kmem_charge 2006-10-05 11:42:43.000000000 +0400
+++ ./include/asm-x86_64/thread_info.h 2006-10-05 12:14:35.000000000 +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.bc_kmem_charge 2006-09-20 14:46:41.000000000 +0400
+++ ./ipc/msgutil.c 2006-10-05 12:14:35.000000000 +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.bc_kmem_charge 2006-10-05 11:42:43.000000000 +0400
+++ ./ipc/sem.c 2006-10-05 12:14:35.000000000 +0400
@@ -1008,7 +1008,7 @@ 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);
@@ -1071,7 +1071,8 @@ 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);
@@ -1132,7 +1133,7 @@ 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.bc_kmem_charge 2006-10-05 11:42:43.000000000 +0400
+++ ./ipc/util.c 2006-10-05 12:14:35.000000000 +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.bc_kmem_charge 2006-10-05 12:11:44.000000000 +0400
+++ ./kernel/fork.c 2006-10-05 12:14:35.000000000 +0400
@@ -138,7 +138,7 @@ 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

  /*
@@ -1433,23 +1433,24 @@ 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.bc_kmem_charge 2006-10-05 11:42:43.000000000 +0400
+++ ./kernel/posix-timers.c 2006-10-05 12:14:35.000000000 +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.bc_kmem_charge 2006-10-05 11:42:43.000000000 +0400
+++ ./kernel/signal.c 2006-10-05 12:14:35.000000000 +0400
@@ -2749,5 +2749,5 @@ 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.bc_kmem_charge 2006-10-05 11:43:39.000000000 +0400
+++ ./kernel/user.c 2006-10-05 12:14:35.000000000 +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.bc_kmem_charge 2006-10-05 11:42:43.000000000 +0400
```

```
+++ ./mm/rmap.c 2006-10-05 12:14:35.000000000 +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.bc_kmem_charge 2006-10-05 11:42:43.000000000 +0400
+++ ./mm/shmem.c 2006-10-05 12:14:35.000000000 +0400
@@ -371,7 +371,8 @@ 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);
--- ./mm/slab.c.bc_kmem_charge 2006-10-05 12:12:24.000000000 +0400
+++ ./mm/slab.c 2006-10-05 12:14:35.000000000 +0400
@@ -1459,7 +1459,8 @@ 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) {
@@ -1467,7 +1468,8 @@ 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);
  }

@@ -1485,7 +1487,8 @@ 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);
   }
 #ifdef CONFIG_ZONE_DMA
```

Subject: [PATCH 8/10] BC: private pages (mappings) accounting (core)
Posted by dev on Thu, 05 Oct 2006 15:54:25 GMT
View Forum Message <> Reply to Message

Accounting of private pages.
Accounting is performed on mmap per vma. When mm changes
its BC, vmas are not recharged.

On mlock and mprotect vma status may change from BC point of
view, so recharging is done accordingly.

See definition of private pages at
http://wiki.openvz.org/User_pages_accounting

Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Signed-off-by: Kirill Korotaev <dev@openvz.org>

```
 include/bc/vmpages.h  |   90 ++++++++++++++++++++++++++++++++++
 include/linux/mm.h    |    3 +
 include/linux/sched.h |    3 +
 kernel/bc/vmpages.c   |  138 ++++++++++++++++++++++++++++++++++++++++++++++++++
 4 files changed, 234 insertions(+)

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/vmpages.h 2006-10-05 12:19:59.000000000 +0400
@@ -0,0 +1,90 @@
+/*
+ * include/bc/vmpages.h
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#ifndef __BC_VMPAGES_H_
+#define __BC_VMPAGES_H_
+
+#include <bc/beancounter.h>
```

```
+
+struct vm_area_struct;
+struct mm_struct;
+struct file;
+
+#define BC_NOCHARGE 0
+#define BC_UNCHARGE 1
+#define BC_CHARGE 2
+
+#ifdef CONFIG_BEANCOUNTERS
+#define __vma_set_bc(vma, bc) do { (vma)->vma_bc = bc_get(bc); } while (0)
+#define vma_set_bc(vma) __vma_set_bc(vma, (vma)->vm_mm->mm_bc)
+#define vma_copy_bc(vma) __vma_set_bc(vma, (vma)->vma_bc)
+#define vma_release_bc(vma) do { bc_put((vma)->vma_bc); } while (0)
+
+#define mm_init_beancounter(mm) do {  \
+  struct beancounter *bc;  \
+  bc = get_exec_bc();  \
+  (mm)->mm_bc = bc_get(bc); \
+ } while (0)
+#define mm_free_beancounter(mm) do { bc_put(mm->mm_bc); } while (0)
+
+int __must_check bc_need_memory_recharge(struct vm_area_struct *vma,
+  struct file *new_file, unsigned long new_flags);
+
+int __must_check __bc_memory_charge(struct mm_struct *mm, unsigned long len,
+  int severity);
+int __must_check bc_memory_charge(struct mm_struct *mm, unsigned long len,
+  struct file *file, unsigned long flags, int severity);
+int __must_check bc_vma_charge(struct vm_area_struct *vma);
+
+void __bc_memory_uncharge(struct mm_struct *mm, unsigned long len);
+void bc_memory_uncharge(struct mm_struct *mm, unsigned long len,
+  struct file *file, unsigned long flags);
+void bc_vma_uncharge(struct vm_area_struct *vma);
+
+#define bc_equal(bc1, bc2) (bc1 == bc2)
+#else
+static inline
+int __must_check bc_need_memory_recharge(struct vm_area_struct *vma,
+  struct file *new_file, unsigned long new_flags)
+{
+ return BC_NOCHARGE;
+}
+static inline int __must_check __bc_memory_charge(struct mm_struct *mm,
+  unsigned long len, int severity)
+{
+ return 0;
```

```
+}
+static inline int __must_check bc_memory_charge(struct mm_struct *mm,
+  unsigned long len, struct file *file, unsigned long flags,
+  int severity)
+{
+ return 0;
+}
+static inline int __must_check bc_vma_charge(struct vm_area_struct *vma)
+{
+ return 0;
+}
+static inline void __bc_memory_uncharge(struct mm_struct *mm, unsigned long len)
+{
+}
+static inline void bc_memory_uncharge(struct mm_struct *mm, unsigned long len,
+  struct file *file, unsigned long flags)
+{
+}
+static inline void bc_vma_uncharge(struct vm_area_struct *vma)
+{
+}
+
+#define mm_init_beancounter(mm) do { } while (0)
+#define mm_free_beancounter(mm) do { } while (0)
+#define __vma_set_bc(vma, bc) do { } while (0)
+#define vma_set_bc(vma)  do { } while (0)
+#define vma_copy_bc(vma) do { } while (0)
+#define vma_release_bc(vma) do { } while (0)
+#define bc_equal(bc1, bc2) 1
+#endif
+#endif
--- ./include/linux/mm.h.bc_vmpages_core 2006-10-05 12:12:24.000000000 +0400
+++ ./include/linux/mm.h 2006-10-05 12:16:03.000000000 +0400
@@ -112,6 +112,9 @@ struct vm_area_struct {
 #ifdef CONFIG_NUMA
  struct mempolicy *vm_policy; /* NUMA policy for the VMA */
 #endif
+#ifdef CONFIG_BEANCOUNTERS
+ struct beancounter *vma_bc;
+#endif
 };

 /*
--- ./include/linux/sched.h.bc_vmpages_core 2006-10-05 12:11:44.000000000 +0400
+++ ./include/linux/sched.h 2006-10-05 12:16:03.000000000 +0400
@@ -369,6 +369,9 @@ struct mm_struct {
  /* aio bits */
  rwlock_t  ioctx_list_lock;
```

```
  struct kioctx  *ioctx_list;
+#ifdef CONFIG_BEANCOUNTERS
+ struct beancounter *mm_bc;
+#endif
 };

 struct sighand_struct {
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/vmpages.c 2006-10-05 12:20:08.000000000 +0400
@@ -0,0 +1,138 @@
+/*
+ * kernel/bc/vmpages.c
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#include <linux/mm.h>
+
+#include <bc/beancounter.h>
+#include <bc/vmpages.h>
+
+#define BC_PRIVVMPAGES_BARRIER BC_MAXVALUE
+#define BC_PRIVVMPAGES_LIMIT BC_MAXVALUE
+
+/*
+ * Core routines
+ */
+
+/*
+ * bc_vma_private checks whether VMA (file, flags) is private
+ * from BC point of view. private VMAs are charged when they are mapped
+ * thus prventing system from resource exhausting when pages from these VMAs
+ * are touched.
+ */
+static inline int bc_vma_private(struct file *file, unsigned long flags)
+{
+ return (flags & VM_LOCKED) ||
+  ((flags & VM_WRITE) && (file == NULL || !(flags & VM_SHARED)));
+}
+
+/*
+ * Accounting is performed in pages (not in Kbytes)
+ */
+static inline int do_memory_charge(struct beancounter *bc,
+  unsigned long len, int severity)
+{
+ return bc_charge(bc, BC_PRIVVMPAGES, len >> PAGE_SHIFT, severity);
```

```
+}
+
+static inline void do_memory_uncharge(struct beancounter *bc, unsigned long len)
+{
+ bc_uncharge(bc, BC_PRIVVMPAGES, len >> PAGE_SHIFT);
+}
+
+/*
+ * API calls
+ */
+
+int __bc_memory_charge(struct mm_struct *mm, unsigned long len, int severity)
+{
+ return do_memory_charge(mm->mm_bc, len, severity);
+}
+
+int bc_memory_charge(struct mm_struct *mm, unsigned long len,
+  struct file *file, unsigned long flags, int severity)
+{
+ int ret;
+
+ ret = 0;
+ if (bc_vma_private(file, flags))
+  ret = do_memory_charge(mm->mm_bc, len, severity);
+ return ret;
+}
+
+int bc_vma_charge(struct vm_area_struct *vma)
+{
+ int ret;
+
+ ret = (bc_vma_private(vma->vm_file, vma->vm_flags) ?
+   do_memory_charge(vma->vm_mm->mm_bc,
+    vma->vm_end - vma->vm_start, BC_BARRIER) : 0);
+ if (ret == 0)
+  vma_set_bc(vma);
+ return ret;
+}
+
+void __bc_memory_uncharge(struct mm_struct *mm, unsigned long len)
+{
+ do_memory_uncharge(mm->mm_bc, len);
+}
+
+void bc_memory_uncharge(struct mm_struct *mm, unsigned long len,
+  struct file *file, unsigned long flags)
+{
+ if (bc_vma_private(file, flags))
```

```
+  do_memory_uncharge(mm->mm_bc, len);
+}
+
+void bc_vma_uncharge(struct vm_area_struct *vma)
+{
+ if (bc_vma_private(vma->vm_file, vma->vm_flags))
+  do_memory_uncharge(vma->vma_bc, vma->vm_end - vma->vm_start);
+ vma_release_bc(vma);
+}
+
+
+int bc_need_memory_recharge(struct vm_area_struct *vma, struct file *new_file,
+  unsigned long new_flags)
+{
+ if (bc_vma_private(vma->vm_file, vma->vm_flags)) {
+  if (bc_vma_private(new_file, new_flags))
+   return BC_NOCHARGE;
+
+  /* private -> non-private */
+  return BC_UNCHARGE;
+ } else {
+  if (!bc_vma_private(new_file, new_flags))
+   return BC_NOCHARGE;
+
+  /* non-private -> private */
+  return BC_CHARGE;
+ }
+}
+
+/*
+ * Generic resource info
+ */
+
+static int bc_privvm_init(struct beancounter *bc, gfp_t mask)
+{
+ bc_init_resource(bc, BC_PRIVVMPAGES,
+   BC_PRIVVMPAGES_BARRIER, BC_PRIVVMPAGES_LIMIT);
+ return 0;
+}
+
+struct bc_resource bc_privvm_resource = {
+ .bcr_name  = "privvmpages",
+ .bcr_init  = bc_privvm_init,
+};
+
+static int __init bc_privvm_init_resource(void)
+{
+ bc_register_resource(BC_PRIVVMPAGES, &bc_privvm_resource);
```

```
+ return 0;
+}
+
+__initcall(bc_privvm_init_resource);
```

Subject: [PATCH 9/10] BC: physical pages accounting/reclamation (core)
Posted by Kirill Korotaev on Thu, 05 Oct 2006 15:55:40 GMT
View Forum Message <> Reply to Message

Accounting of physical pages.
Accounting is performed on page mapping into address space.
If page is shared, it is charged to its first user.

If physpages limit is hit, BC pages are reclaimed on the
same basis as try_to_free_pages() does.

Full BC implementation allows fractions of pages accounting
as well: http://wiki.openvz.org/RSS_fractions_accounting

Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Signed-off-by: Kirill Korotaev <dev@openvz.org>

```
 include/bc/rsspages.h      |   46 +++++++
 include/linux/mm.h         |    1
 include/linux/mm_types.h   |    5
 include/linux/page-flags.h |    4
 kernel/bc/rsspages.c       |  270 ++++++++++++++++++++++++++++++++++++++++++++
 mm/vmscan.c                |  190 ++++++++++++++++++++++++++++++++
 6 files changed, 515 insertions(+), 1 deletion(-)

--- ./include/bc/rsspages.h.bc_rsspages 2006-10-05 12:20:42.000000000 +0400
+++ ./include/bc/rsspages.h 2006-10-05 12:35:57.000000000 +0400
@@ -0,0 +1,46 @@
+/*
+ * include/bc/rsspages.h
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#ifndef __BC_RSSPAGES_H_
+#define __BC_RSSPAGES_H_
+
+#include <linux/compiler.h>
+
```

```
+struct page;
+struct vm_area_struct;
+struct page_beancounter;
+
+#ifdef CONFIG_BEANCOUNTERS
+int __must_check bc_rsspage_prepare(struct page *p,
+  struct vm_area_struct *vma, struct page_beancounter **ppb);
+void bc_rsspage_charge(struct page_beancounter *pb);
+void bc_rsspage_release(struct page_beancounter *pb);
+void bc_rsspage_uncharge(struct page *p);
+
+unsigned long bc_try_to_free_pages(struct beancounter *bc);
+unsigned long bc_isolate_pages(unsigned long nr_to_scan,
+  struct beancounter *bc, struct list_head *dst,
+  int active, unsigned long *scanned);
+unsigned long bc_nr_physpages(struct beancounter *bc);
+#else
+static inline int __must_check bc_rsspage_prepare(struct page *p,
+  struct vm_area_struct *vma, struct page_beancounter **ppb)
+{
+ return 0;
+}
+
+static inline void bc_rsspage_charge(struct page_beancounter *pb)
+{
+}
+static inline void bc_rsspage_release(struct page_beancounter *pb)
+{
+}
+static inline void bc_rsspage_uncharge(struct page *p)
+{
+}
+#endif
+#endif
--- ./include/linux/mm.h.bc_rsspages 2006-10-05 12:16:03.000000000 +0400
+++ ./include/linux/mm.h 2006-10-05 12:36:42.000000000 +0400
@@ -223,6 +223,7 @@ struct mmu_gather;
 struct inode;

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

--- ./include/linux/mm_types.h.bc_rsspages 2006-10-05 12:12:24.000000000 +0400
+++ ./include/linux/mm_types.h 2006-10-05 12:36:21.000000000 +0400
@@ -63,7 +63,10 @@ struct page {
        not kmapped, ie. highmem) */
```

```
 #endif /* WANT_PAGE_VIRTUAL */
 #ifdef CONFIG_BEANCOUNTERS
- struct beancounter *bc;
+ union {
+  struct beancounter *bc;
+  struct page_beancounter *pb;
+ };
 #endif
 #ifdef CONFIG_PAGE_OWNER
  int order;
--- ./include/linux/page-flags.h.bc_rsspages 2006-10-05 11:42:43.000000000 +0400
+++ ./include/linux/page-flags.h 2006-10-05 12:20:42.000000000 +0400
@@ -92,6 +92,7 @@
 #define PG_buddy  19 /* Page is free, on buddy lists */

 #define PG_readahead  20 /* Reminder to do readahead */
+#define PG_charged  21 /* Page is already charged to BC */


 #if (BITS_PER_LONG > 32)
@@ -253,6 +254,9 @@ static inline void SetPageUptodate(struc
 #define SetPageReadahead(page) set_bit(PG_readahead, &(page)->flags)
 #define TestClearPageReadahead(page) test_and_clear_bit(PG_readahead, &(page)->flags)

+#define TestSetPageCharged(page) test_and_set_bit(PG_charged, &(page)->flags)
+#define ClearPageCharged(page) clear_bit(PG_charged, &(page)->flags)
+
 struct page; /* forward declaration */

 int test_clear_page_dirty(struct page *page);
--- ./kernel/bc/rsspages.c.bc_rsspages 2006-10-05 12:20:42.000000000 +0400
+++ ./kernel/bc/rsspages.c 2006-10-05 12:37:40.000000000 +0400
@@ -0,0 +1,270 @@
+/*
+ * kernel/bc/rsspages.c
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#include <linux/mm_types.h>
+#include <linux/mm.h>
+#include <linux/page-flags.h>
+#include <linux/hardirq.h>
+#include <linux/kernel.h>
+
+#include <bc/beancounter.h>
+#include <bc/vmpages.h>
```

```
+#include <bc/rsspages.h>
+
+#include <asm/bitops.h>
+
+#define BC_PHYSPAGES_BARRIER BC_MAXVALUE
+#define BC_PHYSPAGES_LIMIT BC_MAXVALUE
+
+/*
+ * page_beancounter is a tie between page and beancounter page is
+ * charged to. it is used to reclaim pages faster by walking bc's
+ * page list, not zones' ones
+ *
+ * this tie can also be used to implement fractions accounting mechanism
+ * as it is done in OpenVZ kernels
+ */
+struct page_beancounter {
+ struct page *page;
+ struct beancounter *bc;
+ struct list_head list;
+};
+
+/*
+ * API calls
+ */
+
+/*
+ * bc_rsspage_prepare allocates a tie and charges page to vma's beancounter
+ * this must be called in non-atomic context to give a chance for pages
+ * reclaiming. otherwise hitting limits will cause -ENOMEM returned.
+ */
+int bc_rsspage_prepare(struct page *page, struct vm_area_struct *vma,
+  struct page_beancounter **ppb)
+{
+ struct beancounter *bc;
+ struct page_beancounter *pb;
+
+ pb = kmalloc(sizeof(struct page_beancounter), GFP_KERNEL);
+ if (pb == NULL)
+  goto out_nomem;
+
+ bc = vma->vma_bc;
+ if (bc_charge(bc, BC_PHYSPAGES, 1, BC_LIMIT))
+  goto out_charge;
+
+ pb->page = page;
+ pb->bc = bc;
+ *ppb = pb;
+ return 0;
```

```
+
+out_charge:
+ kfree(pb);
+out_nomem:
+ return -ENOMEM;
+}
+
+/*
+ * bc_rsspage_release is a rollback call for bc_rsspage_prepare
+ */
+void bc_rsspage_release(struct page_beancounter *pb)
+{
+ bc_uncharge(pb->bc, BC_PHYSPAGES, 1);
+ kfree(pb);
+}
+
+/*
+ * bc_rsspage_charge actually ties page and beancounter together
+ * and marks page as PG_charged. this is done in not-failing path
+ * to be sure the page IS charged
+ */
+void bc_rsspage_charge(struct page_beancounter *pb)
+{
+ struct page *pg;
+ struct beancounter *bc;
+
+ pg = pb->page;
+ if (TestSetPageCharged(pg)) {
+  bc_rsspage_release(pb);
+  return;
+ }
+
+ bc = bc_get(pb->bc);
+ spin_lock(&bc->bc_page_lock);
+ list_add(&pb->list, &bc->bc_page_list);
+ spin_unlock(&bc->bc_page_lock);
+ page_pb(pg) = pb;
+}
+
+/*
+ * bc_rsspage_uncharge is called when pages is get completely unapped
+ * from all address spaces
+ */
+void bc_rsspage_uncharge(struct page *page)
+{
+ struct page_beancounter *pb;
+ struct beancounter *bc;
+
```

```
+ pb = page_pb(page);
+ if (pb == NULL)
+  return;
+
+ ClearPageCharged(page);
+ page_pb(page) = NULL;
+
+ bc = pb->bc;
+ spin_lock(&bc->bc_page_lock);
+ list_del(&pb->list);
+ spin_unlock(&bc->bc_page_lock);
+
+ bc_uncharge(bc, BC_PHYSPAGES, 1);
+ bc_put(bc);
+ kfree(pb);
+}
+
+/*
+ * Page reclamation helper
+ *
+ * this function resembles isolate_lru_pages() but is scans through
+ * bc's page list, not zone's active/inactive ones.
+ */
+
+unsigned long bc_isolate_pages(unsigned long nr_to_scan, struct beancounter *bc,
+  struct list_head *dst, int active, unsigned long *scanned)
+{
+ unsigned long nr_taken = 0;
+ struct page *page;
+ struct page_beancounter *pb;
+ unsigned long scan;
+ struct list_head *src;
+ LIST_HEAD(pb_list);
+ struct zone *z;
+
+ spin_lock(&bc->bc_page_lock);
+ src = &bc->bc_page_list;
+ for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
+  struct list_head *target;
+  pb = list_entry(src->prev, struct page_beancounter, list);
+  page = pb->page;
+  z = page_zone(page);
+
+  list_move(&pb->list, &pb_list);
+
+  spin_lock_irq(&z->lru_lock);
+  if (PageLRU(page)) {
+   if ((active && PageActive(page)) ||
```

```
+     (!active && !PageActive(page))) {
+    if (likely(get_page_unless_zero(page))) {
+     ClearPageLRU(page);
+     target = dst;
+     nr_taken++;
+     list_move(&page->lru, dst);
+    }
+   }
+  }
+  spin_unlock_irq(&z->lru_lock);
+ }
+
+ list_splice(&pb_list, src);
+ spin_unlock(&bc->bc_page_lock);
+
+ *scanned = scan;
+ return nr_taken;
+}
+
+unsigned long bc_nr_physpages(struct beancounter *bc)
+{
+ return bc->bc_parms[BC_PHYSPAGES].held;
+}
+
+/*
+ * Generic resource info
+ */
+
+static int bc_phys_init(struct beancounter *bc, gfp_t mask)
+{
+ spin_lock_init(&bc->bc_page_lock);
+ INIT_LIST_HEAD(&bc->bc_page_list);
+
+ bc_init_resource(bc, BC_PHYSPAGES,
+   BC_PHYSPAGES_BARRIER, BC_PHYSPAGES_LIMIT);
+ return 0;
+}
+
+static void bc_phys_barrier_hit(struct beancounter *bc)
+{
+ /*
+  * May wake up kswapd here to start asynchronous reclaiming of pages
+  */
+}
+
+static int bc_phys_limit_hit(struct beancounter *bc, unsigned long val,
+  unsigned long flags)
+{
```

```
+ int did_some_progress = 0;
+ struct bc_resource_parm *parm;
+
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+ might_sleep();
+
+ parm = &bc->bc_parms[BC_PHYSPAGES];
+ while (1) {
+  did_some_progress = bc_try_to_free_pages(bc);
+
+  spin_lock_irq(&bc->bc_lock);
+  if (parm->held + val <= parm->limit) {
+   parm->held += val;
+   bc_adjust_maxheld(parm);
+   return 0;
+  }
+
+  if (!did_some_progress) {
+   parm->failcnt++;
+   return -ENOMEM;
+  }
+  spin_unlock_irq(&bc->bc_lock);
+ }
+}
+
+static int bc_phys_change(struct beancounter *bc,
+  unsigned long barrier, unsigned long limit)
+{
+ int did_some_progress;
+ struct bc_resource_parm *parm;
+
+ parm = &bc->bc_parms[BC_PHYSPAGES];
+ if (limit >= parm->held)
+  return 0;
+
+ while (1) {
+  spin_unlock_irq(&bc->bc_lock);
+
+  did_some_progress = bc_try_to_free_pages(bc);
+
+  spin_lock_irq(&bc->bc_lock);
+  if (parm->held < limit)
+   return 0;
+  if (!did_some_progress)
+   return -ENOMEM;
+ }
+}
+
```

```
+struct bc_resource bc_phys_resource = {
+ .bcr_name  = "physpages",
+ .bcr_init  = bc_phys_init,
+ .bcr_change  = bc_phys_change,
+ .bcr_barrier_hit = bc_phys_barrier_hit,
+ .bcr_limit_hit  = bc_phys_limit_hit,
+};
+
+static int __init bc_phys_init_resource(void)
+{
+ bc_register_resource(BC_PHYSPAGES, &bc_phys_resource);
+ return 0;
+}
+
+__initcall(bc_phys_init_resource);
--- ./mm/vmscan.c.bc_rsspages 2006-10-05 11:42:43.000000000 +0400
+++ ./mm/vmscan.c 2006-10-05 12:33:06.000000000 +0400
@@ -38,6 +38,8 @@
 #include <linux/delay.h>
 #include <linux/kthread.h>

+#include <bc/rsspages.h>
+
 #include <asm/tlbflush.h>
 #include <asm/div64.h>

@@ -1076,6 +1078,194 @@ out:
  return ret;
 }

+#ifdef CONFIG_BEANCOUNTERS
+/*
+ * These are bc's inactive and active pages shrinkers.
+ * Thes works like shrink_inactive_list() and shrink_active_list()
+ *
+ * Two main differences is that bc_isolate_pages() is used to isolate
+ * pages, and that reclaim_mapped is considered to be 1 as hitting BC
+ * limit implies we have to shrink _mapped_ pages
+ */
+static unsigned long bc_shrink_pages_inactive(unsigned long max_scan,
+  struct beancounter *bc, struct scan_control *sc)
+{
+ LIST_HEAD(page_list);
+ unsigned long nr_scanned = 0;
+ unsigned long nr_reclaimed = 0;
+
+ do {
+  struct page *page;
```

```
+  unsigned long nr_taken;
+  unsigned long nr_scan;
+  struct zone *z;
+
+  nr_taken = bc_isolate_pages(sc->swap_cluster_max, bc,
+    &page_list, 0, &nr_scan);
+
+  nr_scanned += nr_scan;
+  nr_reclaimed += shrink_page_list(&page_list, sc);
+  if (nr_taken == 0)
+   goto done;
+
+  while (!list_empty(&page_list)) {
+   page = lru_to_page(&page_list);
+   z = page_zone(page);
+
+   spin_lock_irq(&z->lru_lock);
+   VM_BUG_ON(PageLRU(page));
+   SetPageLRU(page);
+   list_del(&page->lru);
+   if (PageActive(page))
+    add_page_to_active_list(z, page);
+   else
+    add_page_to_inactive_list(z, page);
+   spin_unlock_irq(&z->lru_lock);
+
+   put_page(page);
+  }
+   } while (nr_scanned < max_scan);
+done:
+ return nr_reclaimed;
+}
+
+static void bc_shrink_pages_active(unsigned long nr_pages,
+  struct beancounter *bc, struct scan_control *sc)
+{
+ LIST_HEAD(l_hold);
+ LIST_HEAD(l_inactive);
+ LIST_HEAD(l_active);
+ struct page *page;
+ unsigned long nr_scanned;
+ unsigned long nr_deactivated = 0;
+ struct zone *z;
+
+ bc_isolate_pages(nr_pages, bc, &l_hold, 1, &nr_scanned);
+
+ while (!list_empty(&l_hold)) {
+  cond_resched();
```

```
+  page = lru_to_page(&l_hold);
+  list_del(&page->lru);
+  if (page_mapped(page)) {
+   if ((total_swap_pages == 0 && PageAnon(page)) ||
+       page_referenced(page, 0)) {
+    list_add(&page->lru, &l_active);
+    continue;
+   }
+  }
+  nr_deactivated++;
+  list_add(&page->lru, &l_inactive);
+ }
+
+ while (!list_empty(&l_inactive)) {
+  page = lru_to_page(&l_inactive);
+  z = page_zone(page);
+
+  spin_lock_irq(&z->lru_lock);
+  VM_BUG_ON(PageLRU(page));
+  SetPageLRU(page);
+  VM_BUG_ON(!PageActive(page));
+  ClearPageActive(page);
+
+  list_move(&page->lru, &z->inactive_list);
+  z->nr_inactive++;
+  spin_unlock_irq(&z->lru_lock);
+
+  put_page(page);
+ }
+
+ while (!list_empty(&l_active)) {
+  page = lru_to_page(&l_active);
+  z = page_zone(page);
+
+  spin_lock_irq(&z->lru_lock);
+  VM_BUG_ON(PageLRU(page));
+  SetPageLRU(page);
+  VM_BUG_ON(!PageActive(page));
+  list_move(&page->lru, &z->active_list);
+  z->nr_active++;
+  spin_unlock_irq(&z->lru_lock);
+
+  put_page(page);
+ }
+}
+
+/*
+ * This is a reworked shrink_zone() routine - it scans active pages firts,
```

```
+ * then inactive and returns the number of pages reclaimed
+ */
+static unsigned long bc_shrink_pages(int priority, struct beancounter *bc,
+  struct scan_control *sc)
+{
+ unsigned long nr_pages;
+ unsigned long nr_to_scan;
+ unsigned long nr_reclaimed = 0;
+
+ nr_pages = (bc_nr_physpages(bc) >> priority) + 1;
+ if (nr_pages < sc->swap_cluster_max)
+  nr_pages = 0;
+
+ while (nr_pages) {
+  nr_to_scan = min(nr_pages, (unsigned long)sc->swap_cluster_max);
+  nr_pages -= nr_to_scan;
+  bc_shrink_pages_active(nr_to_scan, bc, sc);
+ }
+
+ nr_pages = (bc_nr_physpages(bc) >> priority) + 1;
+ if (nr_pages < sc->swap_cluster_max)
+  nr_pages = 0;
+
+ while (nr_pages) {
+  nr_to_scan = min(nr_pages, (unsigned long)sc->swap_cluster_max);
+  nr_pages -= nr_to_scan;
+  nr_reclaimed += bc_shrink_pages_inactive(nr_to_scan, bc, sc);
+ }
+
+ throttle_vm_writeout();
+ return nr_reclaimed;
+}
+
+/*
+ * This functions works like try_to_free_pages() - it tries
+ * to shrink bc's pages with increasing priority
+ */
+unsigned long bc_try_to_free_pages(struct beancounter *bc)
+{
+ int priority;
+ int ret = 0;
+ unsigned long total_scanned = 0;
+ unsigned long nr_reclaimed = 0;
+ struct scan_control sc = {
+  .gfp_mask = GFP_KERNEL,
+  .may_writepage = !laptop_mode,
+  .swap_cluster_max = SWAP_CLUSTER_MAX,
+  .may_swap = 1,
```

```
+  .swappiness = vm_swappiness,
+ };
+
+ for (priority = DEF_PRIORITY; priority >= 0; priority--) {
+  sc.nr_scanned = 0;
+  nr_reclaimed += bc_shrink_pages(priority, bc, &sc);
+  total_scanned += sc.nr_scanned;
+  if (nr_reclaimed >= sc.swap_cluster_max) {
+   ret = 1;
+   goto out;
+  }
+
+  if (total_scanned > sc.swap_cluster_max +
+     sc.swap_cluster_max / 2) {
+   wakeup_pdflush(laptop_mode ? 0 : total_scanned);
+   sc.may_writepage = 1;
+  }
+
+  if (sc.nr_scanned && priority < DEF_PRIORITY - 2)
+   blk_congestion_wait(WRITE, HZ/10);
+ }
+out:
+ return ret;
+}
+#endif
+
 /*
  * For kswapd, balance_pgdat() will work across all this node's zones until
  * they are all at pages_high.
```

Subject: [PATCH 10/10] BC: user memory accounting (hooks)
Posted by Kirill Korotaev on Thu, 05 Oct 2006 15:56:50 GMT
View Forum Message <> Reply to Message

Install BC hooks in appropriate places for user memory accounting.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Signed-off-by: Kirill Korotaev <dev@openvz.org>

```
 fs/binfmt_elf.c           |    5 +--
 fs/exec.c                 |   14 +++++++-
 include/asm-alpha/mman.h  |    1
 include/asm-generic/mman.h |    1
 include/asm-mips/mman.h   |    1
 include/asm-parisc/mman.h |    1
```

```
 include/asm-xtensa/mman.h  |    1
 kernel/fork.c              |   11 ++++++
 mm/fremap.c                |   10 ++++++
 mm/memory.c                |   40 +++++++++++++++++++++-
 mm/migrate.c               |   10 ++++++
 mm/mlock.c                 |   16 +++++++++
 mm/mmap.c                  |   74 +++++++++++++++++++++++++++++++++++++++++++-------
 mm/mprotect.c              |   18 ++++++++++
 mm/mremap.c                |   22 +++++++++++---
 mm/rmap.c                  |    3 +
 mm/shmem.c                 |   15 +++++++++
 17 files changed, 219 insertions(+), 24 deletions(-)

--- ./fs/binfmt_elf.c.bc_userpages 2006-10-05 11:42:42.000000000 +0400
+++ ./fs/binfmt_elf.c 2006-10-05 12:44:20.000000000 +0400
@@ -360,7 +360,7 @@ static unsigned long load_elf_interp(str
  eppnt = elf_phdata;
  for (i = 0; i < interp_elf_ex->e_phnum; i++, eppnt++) {
   if (eppnt->p_type == PT_LOAD) {
-   int elf_type = MAP_PRIVATE | MAP_DENYWRITE;
+   int elf_type = MAP_PRIVATE|MAP_DENYWRITE|MAP_EXECPRIO;
    int elf_prot = 0;
    unsigned long vaddr = 0;
    unsigned long k, map_addr;
@@ -846,7 +846,8 @@ static int load_elf_binary(struct linux_
   if (elf_ppnt->p_flags & PF_X)
    elf_prot |= PROT_EXEC;

-  elf_flags = MAP_PRIVATE | MAP_DENYWRITE | MAP_EXECUTABLE;
+  elf_flags = MAP_PRIVATE | MAP_DENYWRITE | MAP_EXECUTABLE |
+    MAP_EXECPRIO;

   vaddr = elf_ppnt->p_vaddr;
   if (loc->elf_ex.e_type == ET_EXEC || load_addr_set) {
--- ./fs/exec.c.bc_userpages 2006-10-05 11:42:42.000000000 +0400
+++ ./fs/exec.c 2006-10-05 12:44:20.000000000 +0400
@@ -50,6 +50,8 @@
 #include <linux/cn_proc.h>
 #include <linux/audit.h>

+#include <bc/rsspages.h>
+
 #include <asm/uaccess.h>
 #include <asm/mmu_context.h>

@@ -308,27 +310,35 @@ void install_arg_page(struct vm_area_str
  struct mm_struct *mm = vma->vm_mm;
  pte_t * pte;
```

```
  spinlock_t *ptl;
+ struct page_beancounter *pc;

  if (unlikely(anon_vma_prepare(vma)))
   goto out;

+ if (bc_rsspage_prepare(page, vma, &pc))
+  goto out;
+
  flush_dcache_page(page);
  pte = get_locked_pte(mm, address, &ptl);
  if (!pte)
-  goto out;
+  goto out_unch;
  if (!pte_none(*pte)) {
   pte_unmap_unlock(pte, ptl);
-  goto out;
+  goto out_unch;
  }
  inc_mm_counter(mm, anon_rss);
  lru_cache_add_active(page);
  set_pte_at(mm, address, pte, pte_mkdirty(pte_mkwrite(mk_pte(
      page, vma->vm_page_prot))));
  page_add_new_anon_rmap(page, vma, address);
+ bc_rsspage_charge(pc);
  pte_unmap_unlock(pte, ptl);

  /* no need for flush_tlb */
  return;
+
+out_unch:
+ bc_rsspage_release(pc);
 out:
  __free_page(page);
  force_sig(SIGKILL, current);
--- ./include/asm-alpha/mman.h.bc_userpages 2006-09-20 14:46:38.000000000 +0400
+++ ./include/asm-alpha/mman.h 2006-10-05 12:44:20.000000000 +0400
@@ -14,6 +14,7 @@
 #define MAP_TYPE 0x0f  /* Mask for type of mapping (OSF/1 is _wrong_) */
 #define MAP_FIXED 0x100  /* Interpret addr exactly */
 #define MAP_ANONYMOUS 0x10  /* don't use a file */
+#define MAP_EXECPRIO 0x20  /* charge with BC_LIMIT severity */

 /* not used by linux, but here to make sure we don't clash with OSF/1 defines */
 #define _MAP_HASSEMAPHORE 0x0200
--- ./include/asm-generic/mman.h.bc_userpages 2006-09-20 14:46:38.000000000 +0400
+++ ./include/asm-generic/mman.h 2006-10-05 12:44:20.000000000 +0400
@@ -19,6 +19,7 @@
```

```
 #define MAP_TYPE 0x0f  /* Mask for type of mapping */
 #define MAP_FIXED 0x10  /* Interpret addr exactly */
 #define MAP_ANONYMOUS 0x20  /* don't use a file */
+#define MAP_EXECPRIO 0x2000  /* charge with BC_LIMIT severity */

 #define MS_ASYNC 1  /* sync memory asynchronously */
 #define MS_INVALIDATE 2  /* invalidate the caches */
--- ./include/asm-mips/mman.h.bc_userpages 2006-09-20 14:46:39.000000000 +0400
+++ ./include/asm-mips/mman.h 2006-10-05 12:44:20.000000000 +0400
@@ -46,6 +46,7 @@
 #define MAP_LOCKED 0x8000  /* pages are locked */
 #define MAP_POPULATE 0x10000  /* populate (prefault) pagetables */
 #define MAP_NONBLOCK 0x20000  /* do not block on IO */
+#define MAP_EXECPRIO 0x40000  /* charge with BC_LIMIT severity */

 /*
  * Flags for msync
--- ./include/asm-parisc/mman.h.bc_userpages 2006-09-20 14:46:39.000000000 +0400
+++ ./include/asm-parisc/mman.h 2006-10-05 12:44:20.000000000 +0400
@@ -22,6 +22,7 @@
 #define MAP_GROWSDOWN 0x8000  /* stack-like segment */
 #define MAP_POPULATE 0x10000  /* populate (prefault) pagetables */
 #define MAP_NONBLOCK 0x20000  /* do not block on IO */
+#define MAP_EXECPRIO 0x40000  /* charge with BC_LIMIT severity */

 #define MS_SYNC  1  /* synchronous memory sync */
 #define MS_ASYNC 2  /* sync memory asynchronously */
--- ./include/asm-xtensa/mman.h.bc_userpages 2006-09-20 14:46:40.000000000 +0400
+++ ./include/asm-xtensa/mman.h 2006-10-05 12:44:20.000000000 +0400
@@ -53,6 +53,7 @@
 #define MAP_LOCKED 0x8000  /* pages are locked */
 #define MAP_POPULATE 0x10000  /* populate (prefault) pagetables */
 #define MAP_NONBLOCK 0x20000  /* do not block on IO */
+#define MAP_EXECPRIO 0x40000  /* charge with BC_LIMIT severity */

 /*
  * Flags for msync
--- ./kernel/fork.c.bc_userpages 2006-10-05 12:14:35.000000000 +0400
+++ ./kernel/fork.c 2006-10-05 12:44:20.000000000 +0400
@@ -50,6 +50,7 @@
 #include <linux/random.h>

 #include <bc/task.h>
+#include <bc/vmpages.h>

 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
@@ -252,6 +253,9 @@ static inline int dup_mmap(struct mm_str
```

```
   tmp->vm_flags &= ~VM_LOCKED;
   tmp->vm_mm = mm;
   tmp->vm_next = NULL;
+  vma_set_bc(tmp);
+  if (bc_vma_charge(tmp))
+   goto fail_charge;
   anon_vma_link(tmp);
   file = tmp->vm_file;
   if (file) {
@@ -294,6 +298,10 @@ out:
  flush_tlb_mm(oldmm);
  up_write(&oldmm->mmap_sem);
  return retval;
+
+fail_charge:
+ mpol_free(pol);
+ vma_release_bc(tmp);
 fail_nomem_policy:
  kmem_cache_free(vm_area_cachep, tmp);
 fail_nomem:
@@ -344,6 +352,7 @@ static struct mm_struct * mm_init(struct
  mm->cached_hole_size = ~0UL;

  if (likely(!mm_alloc_pgd(mm))) {
+  mm_init_beancounter(mm);
   mm->def_flags = 0;
   return mm;
  }
@@ -374,6 +383,7 @@ struct mm_struct * mm_alloc(void)
 void fastcall __mmdrop(struct mm_struct *mm)
 {
  BUG_ON(mm == &init_mm);
+ mm_free_beancounter(mm);
  mm_free_pgd(mm);
  destroy_context(mm);
  free_mm(mm);
@@ -508,6 +518,7 @@ fail_nocontext:
   * If init_new_context() failed, we cannot use mmput() to free the mm
   * because it calls destroy_context()
   */
+ mm_free_beancounter(mm);
  mm_free_pgd(mm);
  free_mm(mm);
  return NULL;
--- ./mm/fremap.c.bc_userpages 2006-10-05 11:42:43.000000000 +0400
+++ ./mm/fremap.c 2006-10-05 12:44:20.000000000 +0400
@@ -16,6 +16,8 @@
 #include <linux/module.h>
```

```
 #include <linux/syscalls.h>

+#include <bc/rsspages.h>
+
 #include <asm/mmu_context.h>
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>
@@ -57,6 +59,10 @@ int install_page(struct mm_struct *mm, s
  pte_t *pte;
  pte_t pte_val;
  spinlock_t *ptl;
+ struct page_beancounter *pc;
+
+ if (bc_rsspage_prepare(page, vma, &pc))
+  goto out_nocharge;

  pte = get_locked_pte(mm, addr, &ptl);
  if (!pte)
@@ -84,10 +90,14 @@ int install_page(struct mm_struct *mm, s
  page_add_file_rmap(page);
  update_mmu_cache(vma, addr, pte_val);
  lazy_mmu_prot_update(pte_val);
+ bc_rsspage_charge(pc);
  err = 0;
 unlock:
  pte_unmap_unlock(pte, ptl);
 out:
+ if (err != 0)
+  bc_rsspage_release(pc);
+out_nocharge:
  return err;
 }
 EXPORT_SYMBOL(install_page);
--- ./mm/memory.c.bc_userpages 2006-10-05 11:42:43.000000000 +0400
+++ ./mm/memory.c 2006-10-05 12:44:20.000000000 +0400
@@ -57,6 +57,8 @@
 #include <asm/tlbflush.h>
 #include <asm/pgtable.h>

+#include <bc/rsspages.h>
+
 #include <linux/swapops.h>
 #include <linux/elf.h>

@@ -1483,6 +1485,7 @@ static int do_wp_page(struct mm_struct *
  pte_t entry;
  int reuse = 0, ret = VM_FAULT_MINOR;
  struct page *dirty_page = NULL;
```

```
+ struct page_beancounter *pc;

  old_page = vm_normal_page(vma, address, orig_pte);
  if (!old_page)
@@ -1568,6 +1571,9 @@ gotten:
   cow_user_page(new_page, old_page, address);
  }

+ if (bc_rsspage_prepare(new_page, vma, &pc))
+  goto oom;
+
  /*
   * Re-check the pte - we dropped the lock
   */
@@ -1596,11 +1602,14 @@ gotten:
   update_mmu_cache(vma, address, entry);
   lru_cache_add_active(new_page);
   page_add_new_anon_rmap(new_page, vma, address);
+  bc_rsspage_charge(pc);

   /* Free the old page.. */
   new_page = old_page;
   ret |= VM_FAULT_WRITE;
- }
+ } else
+  bc_rsspage_release(pc);
+
  if (new_page)
   page_cache_release(new_page);
  if (old_page)
@@ -1977,6 +1986,7 @@ static int do_swap_page(struct mm_struct
  swp_entry_t entry;
  pte_t pte;
  int ret = VM_FAULT_MINOR;
+ struct page_beancounter *pc;

  if (!pte_unmap_same(mm, pmd, page_table, orig_pte))
   goto out;
@@ -2009,6 +2019,11 @@ static int do_swap_page(struct mm_struct
   count_vm_event(PGMAJFAULT);
  }

+ if (bc_rsspage_prepare(page, vma, &pc)) {
+  ret = VM_FAULT_OOM;
+  goto out;
+ }
+
  delayacct_clear_flag(DELAYACCT_PF_SWAPIN);
```

```
  mark_page_accessed(page);
  lock_page(page);
@@ -2022,6 +2037,7 @@ static int do_swap_page(struct mm_struct

  if (unlikely(!PageUptodate(page))) {
   ret = VM_FAULT_SIGBUS;
+  bc_rsspage_uncharge(page);
   goto out_nomap;
  }

@@ -2037,6 +2053,7 @@ static int do_swap_page(struct mm_struct
  flush_icache_page(vma, page);
  set_pte_at(mm, address, page_table, pte);
  page_add_anon_rmap(page, vma, address);
+ bc_rsspage_charge(pc);

  swap_free(entry);
  if (vm_swap_full())
@@ -2058,6 +2075,7 @@ unlock:
 out:
  return ret;
 out_nomap:
+ bc_rsspage_release(pc);
  pte_unmap_unlock(page_table, ptl);
  unlock_page(page);
  page_cache_release(page);
@@ -2076,6 +2094,7 @@ static int do_anonymous_page(struct mm_s
  struct page *page;
  spinlock_t *ptl;
  pte_t entry;
+ struct page_beancounter *pc;

  if (write_access) {
   /* Allocate our own private page. */
@@ -2087,15 +2106,22 @@ static int do_anonymous_page(struct mm_s
   if (!page)
    goto oom;

+  if (bc_rsspage_prepare(page, vma, &pc))
+   goto oom_release;
+
   entry = mk_pte(page, vma->vm_page_prot);
   entry = maybe_mkwrite(pte_mkdirty(entry), vma);

   page_table = pte_offset_map_lock(mm, pmd, address, &ptl);
-  if (!pte_none(*page_table))
+  if (!pte_none(*page_table)) {
+   bc_rsspage_release(pc);
```

```
    goto release;
+  }
+
   inc_mm_counter(mm, anon_rss);
   lru_cache_add_active(page);
   page_add_new_anon_rmap(page, vma, address);
+  bc_rsspage_charge(pc);
  } else {
   /* Map the ZERO_PAGE - vm_page_prot is readonly */
   page = ZERO_PAGE(address);
@@ -2121,6 +2147,9 @@ unlock:
 release:
  page_cache_release(page);
  goto unlock;
+
+oom_release:
+ page_cache_release(page);
 oom:
  return VM_FAULT_OOM;
 }
@@ -2150,6 +2179,7 @@ static int do_no_page(struct mm_struct *
  int ret = VM_FAULT_MINOR;
  int anon = 0;
  struct page *dirty_page = NULL;
+ struct page_beancounter *pc;

  pte_unmap(page_table);
  BUG_ON(vma->vm_flags & VM_PFNMAP);
@@ -2205,6 +2235,9 @@ retry:
   }
  }

+ if (bc_rsspage_prepare(new_page, vma, &pc))
+  goto oom;
+
  page_table = pte_offset_map_lock(mm, pmd, address, &ptl);
  /*
   * For a file-backed vma, someone could have truncated or otherwise
@@ -2213,6 +2246,7 @@ retry:
   */
  if (mapping && unlikely(sequence != mapping->truncate_count)) {
   pte_unmap_unlock(page_table, ptl);
+  bc_rsspage_release(pc);
   page_cache_release(new_page);
   cond_resched();
   sequence = mapping->truncate_count;
@@ -2249,8 +2283,10 @@ retry:
    get_page(dirty_page);
```

```
    }
   }
+  bc_rsspage_charge(pc);
  } else {
   /* One of our sibling threads was faster, back out. */
+  bc_rsspage_release(pc);
   page_cache_release(new_page);
   goto unlock;
  }
--- ./mm/migrate.c.bc_userpages 2006-10-05 11:42:43.000000000 +0400
+++ ./mm/migrate.c 2006-10-05 12:44:20.000000000 +0400
@@ -134,6 +134,7 @@ static void remove_migration_pte(struct
  pte_t *ptep, pte;
   spinlock_t *ptl;
  unsigned long addr = page_address_in_vma(new, vma);
+ struct page_beancounter *pc;

  if (addr == -EFAULT)
   return;
@@ -157,6 +158,11 @@ static void remove_migration_pte(struct
    return;
   }

+ if (bc_rsspage_prepare(new, vma, &pc)) {
+  pte_unmap(ptep);
+  return;
+ }
+
   ptl = pte_lockptr(mm, pmd);
   spin_lock(ptl);
  pte = *ptep;
@@ -178,13 +184,17 @@ static void remove_migration_pte(struct
   page_add_anon_rmap(new, vma, addr);
  else
   page_add_file_rmap(new);
+ bc_rsspage_charge(pc);

  /* No need to invalidate - it was non-present before */
  update_mmu_cache(vma, addr, pte);
  lazy_mmu_prot_update(pte);
+ pte_unmap_unlock(ptep, ptl);
+ return;

 out:
  pte_unmap_unlock(ptep, ptl);
+ bc_rsspage_release(pc);
 }
```

```
 /*
--- ./mm/mlock.c.bc_userpages 2006-09-20 14:46:41.000000000 +0400
+++ ./mm/mlock.c 2006-10-05 12:44:20.000000000 +0400
@@ -11,6 +11,7 @@
 #include <linux/mempolicy.h>
 #include <linux/syscalls.h>

+#include <bc/vmpages.h>

 static int mlock_fixup(struct vm_area_struct *vma, struct vm_area_struct **prev,
  unsigned long start, unsigned long end, unsigned int newflags)
@@ -19,12 +20,21 @@ static int mlock_fixup(struct vm_area_st
  pgoff_t pgoff;
  int pages;
  int ret = 0;
-
+ int bc_recharge;
+
+ bc_recharge = BC_NOCHARGE;
  if (newflags == vma->vm_flags) {
   *prev = vma;
   goto out;
  }

+ bc_recharge = bc_need_memory_recharge(vma, vma->vm_file, newflags);
+ if (bc_recharge == BC_CHARGE) {
+  ret = __bc_memory_charge(mm, end - start, BC_BARRIER);
+  if (ret < 0)
+   goto out;
+ }
+
  pgoff = vma->vm_pgoff + ((start - vma->vm_start) >> PAGE_SHIFT);
  *prev = vma_merge(mm, *prev, start, end, newflags, vma->anon_vma,
      vma->vm_file, pgoff, vma_policy(vma));
@@ -48,6 +58,8 @@ static int mlock_fixup(struct vm_area_st
  }

 success:
+ if (bc_recharge == BC_UNCHARGE)
+  __bc_memory_uncharge(mm, end - start);
  /*
   * vm_flags is protected by the mmap_sem held in write mode.
   * It's okay if try_to_unmap_one unmaps a page just after we
@@ -67,6 +79,8 @@ success:

  vma->vm_mm->locked_vm -= pages;
 out:
+ if (ret < 0 && bc_recharge == BC_CHARGE)
```

```
+  __bc_memory_uncharge(mm, end - start);
  if (ret == -ENOMEM)
   ret = -EAGAIN;
  return ret;
--- ./mm/mmap.c.bc_userpages 2006-10-05 11:42:43.000000000 +0400
+++ ./mm/mmap.c 2006-10-05 12:44:20.000000000 +0400
@@ -26,6 +26,8 @@
 #include <linux/mempolicy.h>
 #include <linux/rmap.h>

+#include <bc/vmpages.h>
+
 #include <asm/uaccess.h>
 #include <asm/cacheflush.h>
 #include <asm/tlb.h>
@@ -37,6 +39,7 @@
 static void unmap_region(struct mm_struct *mm,
   struct vm_area_struct *vma, struct vm_area_struct *prev,
   unsigned long start, unsigned long end);
+static unsigned long __do_brk(unsigned long addr, unsigned long len, int prio);

 /*
  * WARNING: the debugging will use recursive algorithms so never enable this
@@ -224,6 +227,8 @@ static struct vm_area_struct *remove_vma
  struct vm_area_struct *next = vma->vm_next;

  might_sleep();
+
+ bc_vma_uncharge(vma);
  if (vma->vm_ops && vma->vm_ops->close)
   vma->vm_ops->close(vma);
  if (vma->vm_file)
@@ -271,7 +276,7 @@ asmlinkage unsigned long sys_brk(unsigne
   goto out;

  /* Ok, looks good - let it rip. */
- if (do_brk(oldbrk, newbrk-oldbrk) != oldbrk)
+ if (__do_brk(oldbrk, newbrk-oldbrk, BC_BARRIER) != oldbrk)
   goto out;
 set_brk:
  mm->brk = brk;
@@ -620,6 +625,7 @@ again:   remove_next = 1 + (end > next->
    fput(file);
   mm->map_count--;
   mpol_free(vma_policy(next));
+  vma_release_bc(next);
   kmem_cache_free(vm_area_cachep, next);
   /*
```

```
    * In mprotect's case 6 (see comments on vma_merge),
@@ -761,15 +767,17 @@ struct vm_area_struct *vma_merge(struct
   */
  if (prev && prev->vm_end == addr &&
       mpol_equal(vma_policy(prev), policy) &&
+    bc_equal(mm->mm_bc, prev->vma_bc) &&
      can_vma_merge_after(prev, vm_flags,
-       anon_vma, file, pgoff)) {
+      anon_vma, file, pgoff)) {
    /*
     * OK, it can.  Can we now merge in the successor as well?
     */
    if (next && end == next->vm_start &&
       mpol_equal(policy, vma_policy(next)) &&
+     bc_equal(mm->mm_bc, next->vma_bc) &&
       can_vma_merge_before(next, vm_flags,
-       anon_vma, file, pgoff+pglen) &&
+       anon_vma, file, pgoff + pglen) &&
       is_mergeable_anon_vma(prev->anon_vma,
             next->anon_vma)) {
        /* cases 1, 6 */
@@ -786,8 +794,9 @@ struct vm_area_struct *vma_merge(struct
   */
  if (next && end == next->vm_start &&
       mpol_equal(policy, vma_policy(next)) &&
+    bc_equal(mm->mm_bc, next->vma_bc) &&
      can_vma_merge_before(next, vm_flags,
-       anon_vma, file, pgoff+pglen)) {
+       anon_vma, file, pgoff + pglen)) {
    if (prev && addr < prev->vm_end) /* case 4 */
      vma_adjust(prev, prev->vm_start,
       addr, prev->vm_pgoff, NULL);
@@ -828,6 +837,7 @@ struct anon_vma *find_mergeable_anon_vma

  if (near->anon_vma && vma->vm_end == near->vm_start &&
       mpol_equal(vma_policy(vma), vma_policy(near)) &&
+    bc_equal(vma->vma_bc, near->vma_bc) &&
      can_vma_merge_before(near, vm_flags,
       NULL, vma->vm_file, vma->vm_pgoff +
       ((vma->vm_end - vma->vm_start) >> PAGE_SHIFT)))
@@ -849,6 +859,7 @@ try_prev:

  if (near->anon_vma && near->vm_end == vma->vm_start &&
       mpol_equal(vma_policy(near), vma_policy(vma)) &&
+    bc_equal(vma->vma_bc, near->vma_bc) &&
      can_vma_merge_after(near, vm_flags,
       NULL, vma->vm_file, vma->vm_pgoff))
    return near->anon_vma;
```

```
@@ -1055,6 +1066,10 @@ munmap_back:
   }
  }

+ if (bc_memory_charge(mm, len, file, vm_flags,
+     flags & MAP_EXECPRIO ? BC_LIMIT : BC_BARRIER))
+  goto charge_error;
+
  /*
   * Can we just expand an old private anonymous mapping?
   * The VM_SHARED test is necessary because shmem_zero_setup
@@ -1083,6 +1098,7 @@ munmap_back:
  vma->vm_page_prot = protection_map[vm_flags &
     (VM_READ|VM_WRITE|VM_EXEC|VM_SHARED)];
  vma->vm_pgoff = pgoff;
+ vma_set_bc(vma);

  if (file) {
   error = -EINVAL;
@@ -1139,6 +1155,7 @@ munmap_back:
    fput(file);
   }
   mpol_free(vma_policy(vma));
+  vma_release_bc(vma);
   kmem_cache_free(vm_area_cachep, vma);
  }
 out:
@@ -1168,6 +1185,8 @@ unmap_and_free_vma:
 free_vma:
  kmem_cache_free(vm_area_cachep, vma);
 unacct_error:
+ bc_memory_uncharge(mm, len, file, vm_flags);
+charge_error:
  if (charged)
   vm_unacct_memory(charged);
  return error;
@@ -1497,12 +1516,16 @@ static int acct_stack_growth(struct vm_a
    return -ENOMEM;
  }

+ if (bc_memory_charge(mm, grow << PAGE_SHIFT,
+     vma->vm_file, vma->vm_flags, BC_LIMIT))
+  goto out_charge;
+
  /*
   * Overcommit..  This must be the final test, as it will
   * update security statistics.
   */
```

```
  if (security_vm_enough_memory(grow))
-  return -ENOMEM;
+  goto out_sec;

  /* Ok, everything looks good - let it rip */
  mm->total_vm += grow;
@@ -1510,6 +1533,11 @@ static int acct_stack_growth(struct vm_a
   mm->locked_vm += grow;
  vm_stat_account(mm, vma->vm_flags, vma->vm_file, grow);
  return 0;
+
+out_sec:
+ bc_memory_uncharge(mm, grow << PAGE_SHIFT, vma->vm_file, vma->vm_flags);
+out_charge:
+ return -ENOMEM;
 }

 #if defined(CONFIG_STACK_GROWSUP) || defined(CONFIG_IA64)
@@ -1743,6 +1771,7 @@ int split_vma(struct mm_struct * mm, str

  /* most fields are the same, copy all, and then fixup */
  *new = *vma;
+ vma_copy_bc(new);

  if (new_below)
   new->vm_end = addr;
@@ -1753,6 +1782,7 @@ int split_vma(struct mm_struct * mm, str

  pol = mpol_copy(vma_policy(vma));
  if (IS_ERR(pol)) {
+  vma_release_bc(new);
   kmem_cache_free(vm_area_cachep, new);
   return PTR_ERR(pol);
  }
@@ -1865,7 +1895,8 @@ static inline void verify_mm_writelocked
  *  anonymous maps.  eventually we may be able to do some
  *  brk-specific accounting here.
  */
-unsigned long do_brk(unsigned long addr, unsigned long len)
+static unsigned long __do_brk(unsigned long addr, unsigned long len,
+  int bc_prio)
 {
  struct mm_struct * mm = current->mm;
  struct vm_area_struct * vma, * prev;
@@ -1925,7 +1956,10 @@ unsigned long do_brk(unsigned long addr,
   return -ENOMEM;

  if (security_vm_enough_memory(len >> PAGE_SHIFT))
```

```
-  return -ENOMEM;
+  goto err_sec;
+
+ if (bc_memory_charge(mm, len, NULL, flags, bc_prio))
+  goto err_ch;

  /* Can we just expand an old private anonymous mapping? */
  if (vma_merge(mm, prev, addr, addr + len, flags,
@@ -1936,10 +1970,8 @@ unsigned long do_brk(unsigned long addr,
   * create a vma struct for an anonymous mapping
   */
  vma = kmem_cache_zalloc(vm_area_cachep, GFP_KERNEL);
- if (!vma) {
-  vm_unacct_memory(len >> PAGE_SHIFT);
-  return -ENOMEM;
- }
+ if (!vma)
+  goto err_alloc;

  vma->vm_mm = mm;
  vma->vm_start = addr;
@@ -1948,6 +1980,7 @@ unsigned long do_brk(unsigned long addr,
  vma->vm_flags = flags;
  vma->vm_page_prot = protection_map[flags &
     (VM_READ|VM_WRITE|VM_EXEC|VM_SHARED)];
+ vma_set_bc(vma);
  vma_link(mm, vma, prev, rb_link, rb_parent);
 out:
  mm->total_vm += len >> PAGE_SHIFT;
@@ -1956,8 +1989,19 @@ out:
   make_pages_present(addr, addr + len);
  }
  return addr;
+
+err_alloc:
+ bc_memory_uncharge(mm, len, NULL, flags);
+err_ch:
+ vm_unacct_memory(len >> PAGE_SHIFT);
+err_sec:
+ return -ENOMEM;
 }

+unsigned long do_brk(unsigned long addr, unsigned long len)
+{
+ return __do_brk(addr, len, BC_LIMIT);
+}
 EXPORT_SYMBOL(do_brk);
```

```
 /* Release all mmaps. */
@@ -2019,6 +2063,12 @@ int insert_vm_struct(struct mm_struct *
  if ((vma->vm_flags & VM_ACCOUNT) &&
       security_vm_enough_memory(vma_pages(vma)))
   return -ENOMEM;
+
+ if (bc_vma_charge(vma)) {
+  if (vma->vm_flags & VM_ACCOUNT)
+   vm_unacct_memory(vma_pages(vma));
+  return -ENOMEM;
+ }
  vma_link(mm, vma, prev, rb_link, rb_parent);
  return 0;
 }
@@ -2058,8 +2108,10 @@ struct vm_area_struct *copy_vma(struct v
   new_vma = kmem_cache_alloc(vm_area_cachep, SLAB_KERNEL);
   if (new_vma) {
    *new_vma = *vma;
+   vma_copy_bc(new_vma);
    pol = mpol_copy(vma_policy(vma));
    if (IS_ERR(pol)) {
+    vma_release_bc(new_vma);
     kmem_cache_free(vm_area_cachep, new_vma);
     return NULL;
    }
--- ./mm/mprotect.c.bc_userpages 2006-10-05 11:42:43.000000000 +0400
+++ ./mm/mprotect.c 2006-10-05 12:44:20.000000000 +0400
@@ -22,6 +22,7 @@
 #include <linux/module.h>
 #include <linux/swap.h>
 #include <linux/swapops.h>
+#include <bc/vmpages.h>
 #include <asm/uaccess.h>
 #include <asm/pgtable.h>
 #include <asm/cacheflush.h>
@@ -140,12 +141,20 @@ mprotect_fixup(struct vm_area_struct *vm
  pgoff_t pgoff;
  int error;
  int dirty_accountable = 0;
+ int bc_recharge;

  if (newflags == oldflags) {
   *pprev = vma;
   return 0;
  }

+ bc_recharge = bc_need_memory_recharge(vma, vma->vm_file, newflags);
+ if (bc_recharge == BC_CHARGE) {
```

```
+  error = __bc_memory_charge(mm, end - start, BC_BARRIER);
+  if (error < 0)
+   goto fail_charge;
+ }
+
  /*
   * If we make a private mapping writable we increase our commit;
   * but (without finer accounting) cannot reduce our commit if we
@@ -157,8 +166,9 @@ mprotect_fixup(struct vm_area_struct *vm
  if (newflags & VM_WRITE) {
   if (!(oldflags & (VM_ACCOUNT|VM_WRITE|VM_SHARED))) {
    charged = nrpages;
+   error = -ENOMEM;
    if (security_vm_enough_memory(charged))
-    return -ENOMEM;
+    goto fail_acct;
    newflags |= VM_ACCOUNT;
   }
  }
@@ -189,6 +199,8 @@ mprotect_fixup(struct vm_area_struct *vm
  }

 success:
+ if (bc_recharge == BC_UNCHARGE)
+  __bc_memory_uncharge(mm, end - start);
  /*
   * vm_flags and vm_page_prot are protected by the mmap_sem
   * held in write mode.
@@ -212,6 +224,10 @@ success:

 fail:
  vm_unacct_memory(charged);
+fail_acct:
+ if (bc_recharge == BC_CHARGE)
+  __bc_memory_uncharge(mm, end - start);
+fail_charge:
  return error;
 }
 /*
--- ./mm/mremap.c.bc_userpages 2006-10-05 11:42:43.000000000 +0400
+++ ./mm/mremap.c 2006-10-05 12:44:20.000000000 +0400
@@ -19,6 +19,8 @@
 #include <linux/security.h>
 #include <linux/syscalls.h>

+#include <bc/vmpages.h>
+
 #include <asm/uaccess.h>
```

```
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>
@@ -344,10 +346,16 @@ unsigned long do_mremap(unsigned long ad
   goto out;
  }

+ if (bc_memory_charge(mm, new_len - old_len,
+     vma->vm_file, vma->vm_flags, BC_BARRIER)) {
+  ret = -ENOMEM;
+  goto out;
+ }
+
  if (vma->vm_flags & VM_ACCOUNT) {
   charged = (new_len - old_len) >> PAGE_SHIFT;
   if (security_vm_enough_memory(charged))
-   goto out_nc;
+    goto out_bc;
  }

  /* old_len exactly to the end of the area..
@@ -374,7 +382,7 @@ unsigned long do_mremap(unsigned long ad
          addr + new_len);
    }
    ret = addr;
-   goto out;
+    goto out_nc;
   }
  }

@@ -393,14 +401,18 @@ unsigned long do_mremap(unsigned long ad
       vma->vm_pgoff, map_flags);
    ret = new_addr;
    if (new_addr & ~PAGE_MASK)
-    goto out;
+     goto out_nc;
   }
   ret = move_vma(vma, addr, old_len, new_len, new_addr);
  }
-out:
+out_nc:
  if (ret & ~PAGE_MASK)
   vm_unacct_memory(charged);
-out_nc:
+out_bc:
+ if (ret & ~PAGE_MASK)
+  bc_memory_uncharge(mm, new_len - old_len,
+     vma->vm_file, vma->vm_flags);
+out:
```

```
  return ret;
 }

--- ./mm/rmap.c.bc_userpages 2006-10-05 12:14:35.000000000 +0400
+++ ./mm/rmap.c 2006-10-05 12:44:20.000000000 +0400
@@ -54,6 +54,8 @@
 #include <linux/rcupdate.h>
 #include <linux/module.h>

+#include <bc/rsspages.h>
+
 #include <asm/tlbflush.h>

 struct kmem_cache *anon_vma_cachep;
@@ -586,6 +588,7 @@ void page_remove_rmap(struct page *page)
   }
 #endif
   BUG_ON(page_mapcount(page) < 0);
+  bc_rsspage_uncharge(page);
   /*
    * It would be tidy to reset the PageAnon mapping here,
    * but that might overwrite a racing page_add_anon_rmap
--- ./mm/shmem.c.bc_userpages 2006-10-05 12:14:35.000000000 +0400
+++ ./mm/shmem.c 2006-10-05 12:44:20.000000000 +0400
@@ -49,6 +49,8 @@
 #include <linux/migrate.h>
 #include <linux/highmem.h>

+#include <bc/vmpages.h>
+
 #include <asm/uaccess.h>
 #include <asm/div64.h>
 #include <asm/pgtable.h>
@@ -2511,6 +2513,19 @@ int shmem_zero_setup(struct vm_area_stru

  if (vma->vm_file)
   fput(vma->vm_file);
+ else {
+  switch (bc_need_memory_recharge(vma, file, vma->vm_flags)) {
+  case BC_UNCHARGE:
+   __bc_memory_uncharge(vma->vm_mm,
+     vma->vm_end - vma->vm_start);
+   break;
+  case BC_CHARGE:
+   if (__bc_memory_charge(vma->vm_mm,
+      vma->vm_end - vma->vm_start,
+      BC_BARRIER))
+    return -ENOMEM;
```

```
+ }
+ }
  vma->vm_file = file;
  vma->vm_ops = &shmem_vm_ops;
  return 0;
```

Subject: [TOOL] BC: bcctl.c
Posted by Kirill Korotaev on Thu, 05 Oct 2006 15:59:20 GMT
View Forum Message <> Reply to Message

Utility (bcctl) usage and examples:
Usage: bcctl <cmd> [args...]
Commands:
    get
        gets current bc id
    enter <id> [<pid>]
        moves task into bc.
        if no pid - moves parent task
    stat <id> [<resource>]
        prints stats for bc's resource.
        if no resource - prints all resources
    set <id> <resource> <barrier> [<limit>]
        sets barrier/limit for bc's resource
        if no limit - it is set equal to barrier
Resources:
    kmemsize
    privvmpages
    physpages

Examples:
Move shell into beancounter
  $ bcctl enter 1
Look statistics
  $ bcctl stat 1
  resource   held   minheld   maxheld   barrier   limit   failcnt
  kmemsize   824   0   1056   65536   65536   0
  privvmpages   100   0   304 2147483647 2147483647   0
  physpages   40   0   150 2147483647 2147483647   0
See how reclamation work
  $ bcctl set 1 physpages 200
  $ memeat
  $ bcctl stat 1 physpages
  resource   held   minheld   maxheld   barrier   limit   failcnt
  physpages   40   0   200   200   200   0
Now turn swap off and run memeater again
  $ swapoff -a
  $ memeat

```
 Killed
 $ bcctl stat 1 physpages
 resource          held    minheld    maxheld    barrier      limit   failcnt
 physpages           41          0        200        200        200         1
```

```
#include <sys/syscall.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

typedef unsigned int bcid_t;

#include "bc/beancounter.h"

#define get_bcid()  (syscall(__NR_get_bcid))
#define set_bcid(id, pid) (syscall(__NR_set_bcid, id, pid))
#define set_bclimit(id, res, l) (syscall(__NR_set_bclimit, id, res, l))
#define get_bcstat(id, res, s) (syscall(__NR_get_bcstat, id, res, s))

static char *res_names[] = {
 "kmemsize",
 "privvmpages",
 "physpages",
 NULL,
};

static int bc_get_id(int argc, char **argv)
{
 bcid_t id;

 id = get_bcid();
 printf("%u\n", id);
 return 0;
}

static int bc_enter_id(int argc, char **argv)
{
 bcid_t id;
 pid_t pid;
 char *end;
 int ret;

 if (argc < 1)
  return -2;
```

```
 id = strtol(argv[0], &end, 10);
 if (*end != '\0')
  return -2;

 if (argc > 1) {
  pid = strtol(argv[1], &end, 10);
  if (*end != '\0')
   return -2;
 } else
  pid = getppid();

 ret = set_bcid(id, pid);
 if (ret < 0)
  perror("Can't set bcid");
 return ret;
}

static int bc_set_parms(int argc, char **argv)
{
 bcid_t id;
 int res;
 unsigned long limits[2];
 char *end;
 int ret;

 if (argc < 3)
  return -2;

 id = strtol(argv[0], &end, 10);
 if (*end != '\0')
  return -2;

 for (res = 0; res_names[res] != NULL; res++)
  if (strcmp(res_names[res], argv[1]) == 0)
   break;
 if (res_names[res] == NULL)
  return -2;

 limits[0] = strtoul(argv[2], &end, 10);
 if (*end != '\0')
  return -2;

 if (argc > 3) {
  limits[1] = strtoul(argv[3], &end, 10);
  if (*end != '\0')
   return -2;
 } else
  limits[1] = limits[0];
```

```
 ret = set_bclimit(id, res, limits);
 if (ret < 0)
  perror("Can't set limits");
 return ret;
}

static void bc_show_stat_hdr(void)
{
 printf("%-12s %10s %10s %10s %10s %10s %10s\n",
   "resource",
   "held",
   "minheld",
   "maxheld",
   "barrier",
   "limit",
   "failcnt");
}

static int bc_show_stat(bcid_t id, int res)
{
 struct bc_resource_parm parm;
 int ret;

 ret = get_bcstat(id, res, &parm);
 if (ret < 0) {
  perror("Can't get stat");
  return ret;
 }

 printf("%-12s %10lu %10lu %10lu %10lu %10lu %10lu\n",
   res_names[res],
   parm.held,
   parm.minheld,
   parm.maxheld,
   parm.barrier,
   parm.limit,
   parm.failcnt);
 return 0;
}

static int bc_get_stat(int argc, char **argv)
{
 bcid_t id;
 int res, ret;
 char *end;

 if (argc < 1)
```

```
  return -2;

 id = strtol(argv[0], &end, 10);
 if (*end != '\0')
  return -2;

 if (argc > 1) {
  for (res = 0; res_names[res] != NULL; res++)
   if (strcmp(res_names[res], argv[1]) == 0)
    break;
  if (res_names[res] == NULL)
   return -2;

  bc_show_stat_hdr();
  return bc_show_stat(id, res);
 }

 bc_show_stat_hdr();
 for (res = 0; res_names[res] != NULL; res++) {
  ret = bc_show_stat(id, res);
  if (ret < 0)
   return ret;
 }
 return 0;
}

static struct bc_command {
 char *name;
 int (*cmd)(int argc, char **argv);
 char *args;
 char *help;
} bc_commands[] = {
 {
  .name = "get",
  .cmd = bc_get_id,
  .args = "",
  .help = "gets current bc id",
 },
 {
  .name = "enter",
  .cmd = bc_enter_id,
  .args = "<id> [<pid>]",
  .help = "moves task into bc.\n"
    "\t\tif no pid - moves parent task",
 },
 {
  .name = "stat",
  .cmd = bc_get_stat,
```

```
  .args = "<id> [<resource>]",
  .help = "prints stats for bc's resource.\n\t\t"
    "if no resource - prints all resources",
 },
 {
  .name = "set",
  .cmd = bc_set_parms,
  .args = "<id> <resource> <barrier> [<limit>]",
  .help = "sets barrier/limit for bc's resource\n"
    "\t\tif no limit - it is set equal to barrier",
 },
 {
  .name = NULL,
 },
};

static void usage(void)
{
 int i;

 printf("Usage: bcctl <cmd> [args...]\n");
 printf("Commands:\n");
 for (i = 0; bc_commands[i].name != NULL; i++)
  printf("\t%s %s\n\t\t%s\n",
    bc_commands[i].name,
    bc_commands[i].args,
    bc_commands[i].help);
 printf("Resources:\n");
 for (i = 0; res_names[i] != NULL; i++)
  printf("\t%s\n", res_names[i]);
}

int main(int argc, char **argv)
{
 int i, ret;

 if (argc < 2) {
  usage();
  return 0;
 }

 ret = -1;
 for (i = 0; bc_commands[i].name != NULL; i++) {
  if (strcmp(argv[1], bc_commands[i].name) != 0)
   continue;

  ret = bc_commands[i].cmd(argc - 2, argv + 2);
 }
```

```
 if (ret < 0)
  usage();
 return ret;
}
```