Subject: BC: resource beancounters (v6) (with userpages reclamation + configfs)
Posted by dev on Thu, 09 Nov 2006 16:42:56 GMT
View Forum Message <> Reply to Message

MAJOR CHANGES in v6 (see details below):
- configfs interface instead of syscalls (as wanted by CKRM people...)
- added numfiles resource accounting
- added numtasks resource accounting

numfiles and numtasks controllers demonstrate how
clean and simple BC interface is.

Patch set is applicable to 2.6.19-rc5-mm1

Resource BeanCounters (BC).

BC allows to account and control consumption
of kernel resources used by *group* of processes
(users, containers, ...).

Draft BC 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 and not reclaimable 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)

Summary of changes from v5 patch set:

* configfs interface instead of syscalls (as wanted by CKRM people)
* added numfiles resource accounting
* added numtasks resource accounting
* introduced dummy_resource to handle case when
  no resource registered
* calls to rss accounting are integrated to rmap calls

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.

Thanks,
Kirill

Subject: [PATCH 1/13] BC: atomic_dec_and_lock_irqsave() helper
Posted by dev on Thu, 09 Nov 2006 16:47:20 GMT
View Forum Message <> Reply to Message

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

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

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

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

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

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

```
--- ./include/linux/spinlock.h.bcprep 2006-11-03 17:46:25.000000000 +0300
+++ ./include/linux/spinlock.h 2006-11-03 17:46:31.000000000 +0300
@@ -319,6 +319,12 @@ 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.bcprep 2006-11-03 17:46:25.000000000 +0300
+++ ./kernel/user.c 2006-11-03 17:46:31.000000000 +0300
@@ -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.bcprep 2006-11-03 17:46:25.000000000 +0300
+++ ./lib/dec_and_lock.c 2006-11-03 17:46:31.000000000 +0300
@@ -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 2/13] BC: Kconfig and Makefile
Posted by dev on Thu, 09 Nov 2006 16:47:51 GMT
View Forum Message <> Reply to Message

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

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

```
 init/Kconfig       |    4 ++++
 kernel/Makefile    |    1 +
 kernel/bc/Kconfig  |   17 ++++++++++++++++
 kernel/bc/Makefile |   11 ++++++++++
 4 files changed, 33 insertions(+)

--- ./init/Kconfig.bckconfig 2006-11-09 11:29:12.000000000 +0300
+++ ./init/Kconfig 2006-11-09 11:30:21.000000000 +0300
@@ -585,6 +585,10 @@ config STOP_MACHINE
    Need stop_machine() primitive.
 endmenu

+menu "Beancounters"
+source "kernel/bc/Kconfig"
+endmenu
+
 menu "Block layer"
 source "block/Kconfig"
 endmenu
--- ./kernel/Makefile.bckconfig 2006-11-09 11:29:12.000000000 +0300
+++ ./kernel/Makefile 2006-11-09 11:30:21.000000000 +0300
@@ -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 ($(CONFIG_PROC_FS),y)
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/Kconfig 2006-11-09 11:30:21.000000000 +0300
@@ -0,0 +1,17 @@
+config BEANCOUNTERS
+ bool "Enable resource accounting/control"
+ default n
+ depends on CONFIGFS_FS
+ help
+   When Y this option provides accounting and allows configuring
+   limits for user's consumption of exhaustible system resources.
+   The most important resource controlled by this patch is unswappable
+   memory (either mlock'ed or used by internal kernel structures and
+   buffers). The main goal of this patch is to protect processes
+   from running short of important resources because of accidental
+   misbehavior of processes or malicious activity aiming to ``kill''
+   the system. It's worth mentioning that resource limits configured
+   by setrlimit(2) do not give an acceptable level of protection
+   because they cover only a small fraction of resources and work on a
+   per-process basis.  Per-process accounting doesn't prevent malicious
+   users from spawning a lot of resource-consuming processes.
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/Makefile 2006-11-09 11:31:24.000000000 +0300
@@ -0,0 +1,11 @@
+#
+# kernel/bc/Makefile
+#
+# Copyright (C) 2006 OpenVZ SWsoft Inc.
+#
+
+obj-y = beancounter.o vmpages.o rsspages.o kmem.o misc.o
+
+obj-$(CONFIG_CONFIGFS_FS) += bc_if.o
+
+bc_if-objs := configfs.o
```

Subject: [PATCH 3/13] BC: beancounters core and API
Posted by dev on Thu, 09 Nov 2006 16:49:24 GMT

View Forum Message <> Reply to Message

Core functionality and interfaces of BC:
find/create beancounter, initialization,
charge/uncharge of resource, core objects' declarations.

Basic structures:

bc_resource_parm - resource description
beancounter - set of resources, id, lock

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

---

include/bc/beancounter.h | 171 ++++++

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

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

```
+
+void __must_check bc_uncharge_locked(struct beancounter *bc, int res_id,
+  unsigned long val);
+static inline void bc_uncharge(struct beancounter *bc, int res_id,
+  unsigned long val)
+{
+ unsigned long flags;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ bc_uncharge_locked(bc, res_id, val);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+}
+
+void __init bc_register_resource(int res_id, struct bc_resource *br);
+void __init bc_init_early(void);
+#else /* CONFIG_BEANCOUNTERS */
+static inline int __must_check bc_charge_locked(struct beancounter *bc, int res,
+  unsigned long val, int strict, unsigned long flags)
+{
+ return 0;
+}
+
+static inline int __must_check bc_charge(struct beancounter *bc, int res,
+  unsigned long val, int strict)
+{
+ return 0;
+}
+
+static inline void bc_uncharge_locked(struct beancounter *bc, int res,
+  unsigned long val)
+{
+}
+
+static inline void bc_uncharge(struct beancounter *bc, int res,
+  unsigned long val)
+{
+}
+
+static inline void bc_init_early(void)
+{
+}
+#endif /* CONFIG_BEANCOUNTERS */
+#endif /* __KERNEL__ */
+#endif
--- ./include/linux/types.h.bcprep 2006-11-03 17:46:25.000000000 +0300
+++ ./include/linux/types.h 2006-11-03 17:46:31.000000000 +0300
@@ -40,6 +40,21 @@ typedef __kernel_gid32_t gid_t;
 typedef __kernel_uid16_t        uid16_t;
```

```
 typedef __kernel_gid16_t        gid16_t;

+/*
+ * Type of beancounter id (CONFIG_BEANCOUNTERS)
+ *
+ * The ancient Unix implementations of this kind of resource management and
+ * security are built around setluid() which sets a uid value that cannot
+ * be changed again and is normally used for security purposes. That
+ * happened to be a uid_t and in simple setups at login uid = luid = euid
+ * would be the norm.
+ *
+ * Thus the Linux one happens to be a uid_t. It could be something else but
+ * for the "container per user" model whatever a container is must be able
+ * to hold all possible uid_t values. Alan Cox.
+ */
+typedef uid_t   bcid_t;
+
 #ifdef CONFIG_UID16
 /* This is defined by include/asm-{arch}/posix_types.h */
 typedef __kernel_old_uid_t old_uid_t;
@@ -52,6 +67,7 @@ typedef __kernel_old_gid_t old_gid_t;
 #else
 typedef __kernel_uid_t  uid_t;
 typedef __kernel_gid_t  gid_t;
+typedef __kernel_uid_t  bcid_t;
 #endif /* __KERNEL__ */

 #if defined(__GNUC__) && !defined(__STRICT_ANSI__)
--- ./init/main.c.bccore 2006-11-03 17:46:10.000000000 +0300
+++ ./init/main.c 2006-11-03 17:47:18.000000000 +0300
@@ -53,6 +53,8 @@
 #include <linux/lockdep.h>
 #include <linux/pid_namespace.h>

+#include <bc/beancounter.h>
+
 #include <asm/io.h>
 #include <asm/bugs.h>
 #include <asm/setup.h>
@@ -483,6 +485,7 @@ asmlinkage void __init start_kernel(void
  char * command_line;
  extern struct kernel_param __start___param[], __stop___param[];

+ bc_init_early();
  smp_setup_processor_id();

  /*
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
```

```
+++ ./kernel/bc/beancounter.c 2006-11-03 17:47:18.000000000 +0300
@@ -0,0 +1,253 @@
+/*
+ * kernel/bc/beancounter.c
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#include <linux/sched.h>
+#include <linux/list.h>
+#include <linux/hash.h>
+#include <linux/gfp.h>
+#include <linux/slab.h>
+#include <linux/module.h>
+
+#include <bc/beancounter.h>
+
+#define BC_HASH_BITS (8)
+#define BC_HASH_SIZE (1 << BC_HASH_BITS)
+
+static int bc_dummy_init(struct beancounter *bc, int i)
+{
+ bc_init_resource(&bc->bc_parms[i], BC_MAXVALUE, BC_MAXVALUE);
+ return 0;
+}
+
+static struct bc_resource bc_dummy_res = {
+ .bcr_name = "dummy",
+ .bcr_init = bc_dummy_init,
+};
+
+struct bc_resource *bc_resources[BC_RESOURCES] = {
+ [0 ... BC_RESOURCES - 1] = &bc_dummy_res,
+};
+
+struct beancounter init_bc;
+static struct hlist_head bc_hash[BC_HASH_SIZE];
+static spinlock_t bc_hash_lock;
+static kmem_cache_t *bc_cache;
+
+static void init_beancounter_struct(struct beancounter *bc, bcid_t bcid)
+{
+ bc->bc_id = bcid;
+ spin_lock_init(&bc->bc_lock);
+ atomic_set(&bc->bc_refcount, 1);
+}
+
```

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

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

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

```
+ int ret;
+
+ ret = -EINVAL;
+ if (bar > lim)
+  goto out;
+ if (bar > BC_MAXVALUE || lim > BC_MAXVALUE)
+  goto out;
+
+ ret = 0;
+ spin_lock_irq(&bc->bc_lock);
+ if (bc_resources[res]->bcr_change) {
+  ret = bc_resources[res]->bcr_change(bc, bar, lim);
+  if (ret < 0)
+   goto out_unlock;
+ }
+
+ bc->bc_parms[res].barrier = bar;
+ bc->bc_parms[res].limit = lim;
+
+out_unlock:
+ spin_unlock_irq(&bc->bc_lock);
+out:
+ return ret;
+}
+
+void __init bc_register_resource(int res_id, struct bc_resource *br)
+{
+ BUG_ON(bc_resources[res_id] != &bc_dummy_res);
+ BUG_ON(res_id >= BC_RESOURCES);
+
+ bc_resources[res_id] = br;
+}
+
+void __init bc_init_early(void)
+{
+ int i;
+
+ init_beancounter_struct(&init_bc, 0);
+
+ for (i = 0; i < BC_RESOURCES; i++) {
+  init_bc.bc_parms[i].barrier = BC_MAXVALUE;
+  init_bc.bc_parms[i].limit = BC_MAXVALUE;
+ }
+
+ spin_lock_init(&bc_hash_lock);
+ hlist_add_head(&init_bc.bc_hash, &bc_hash[hash_long(0, BC_HASH_BITS)]);
+}
+
```

```
+int __init bc_init_late(void)
+{
+ bc_cache = kmem_cache_create("beancounters",
+    sizeof(struct beancounter), 0,
+    SLAB_HWCACHE_ALIGN | SLAB_PANIC, NULL, NULL);
+ return 0;
+}
+
+__initcall(bc_init_late);
+
+EXPORT_SYMBOL(bc_resources);
+EXPORT_SYMBOL(init_bc);
+EXPORT_SYMBOL(bc_change_param);
+EXPORT_SYMBOL(bc_findcreate);
+EXPORT_SYMBOL(bc_put);
```

Subject: [PATCH 4/13] BC: context handling
Posted by dev on Thu, 09 Nov 2006 16:51:08 GMT
View Forum Message <> Reply to Message

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

Task references beancounter by exec_bc pointer:
exec_bc: current context. All resources are
        charged to this beancounter.

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

```
 arch/i386/kernel/irq.c  |    5 ++
 fs/proc/array.c         |    6 ++
 include/bc/task.h       |   53 ++++++++++++++++++++++++
 include/linux/sched.h   |    5 ++
 kernel/bc/beancounter.c |    2
 kernel/bc/misc.c        |  100 ++++++++++++++++++++++++++++++++++++++++++++++++
 kernel/fork.c           |   18 +++++++-
 kernel/irq/handle.c     |    6 ++
 kernel/softirq.c        |    5 ++
 9 files changed, 198 insertions(+), 2 deletions(-)

--- ./arch/i386/kernel/irq.c.bcctx 2006-11-09 11:29:09.000000000 +0300
+++ ./arch/i386/kernel/irq.c 2006-11-09 11:32:24.000000000 +0300
@@ -21,6 +21,8 @@
 #include <asm/apic.h>
```

```
 #include <asm/uaccess.h>

+#include <bc/task.h>
+
 DEFINE_PER_CPU(irq_cpustat_t, irq_stat) ____cacheline_internodealigned_in_smp;
 EXPORT_PER_CPU_SYMBOL(irq_stat);

@@ -75,6 +77,7 @@ fastcall unsigned int do_IRQ(struct pt_r
  union irq_ctx *curctx, *irqctx;
  u32 *isp;
 #endif
+ struct beancounter *bc;

  if (unlikely((unsigned)irq >= NR_IRQS)) {
   printk(KERN_EMERG "%s: cannot handle IRQ %d\n",
@@ -99,6 +102,7 @@ fastcall unsigned int do_IRQ(struct pt_r
  }
 #endif

+ bc = set_exec_bc(&init_bc);
 #ifdef CONFIG_4KSTACKS

  curctx = (union irq_ctx *) current_thread_info();
@@ -139,6 +143,7 @@ fastcall unsigned int do_IRQ(struct pt_r
 #endif
   desc->handle_irq(irq, desc);

+ reset_exec_bc(bc, &init_bc);
  irq_exit();
  set_irq_regs(old_regs);
  return 1;
--- ./fs/proc/array.c.bcctx 2006-11-09 11:29:11.000000000 +0300
+++ ./fs/proc/array.c 2006-11-09 11:32:24.000000000 +0300
@@ -76,6 +76,8 @@
 #include <linux/rcupdate.h>
 #include <linux/delayacct.h>

+#include <bc/beancounter.h>
+
 #include <asm/uaccess.h>
 #include <asm/pgtable.h>
 #include <asm/io.h>
@@ -180,6 +182,10 @@ static inline char * task_state(struct t
   p->uid, p->euid, p->suid, p->fsuid,
   p->gid, p->egid, p->sgid, p->fsgid);

+#ifdef CONFIG_BEANCOUNTERS
+ buffer += sprintf(buffer, "Bcid:\t%d\n", p->exec_bc->bc_id);
```

```
+#endif
+
  task_lock(p);
  if (p->files)
   fdt = files_fdtable(p->files);
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/task.h 2006-11-09 11:32:24.000000000 +0300
@@ -0,0 +1,53 @@
+/*
+ * include/bc/task.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;
+
+int __must_check copy_beancounter(struct task_struct *tsk,
+  struct task_struct *parent);
+void free_beancounter(struct task_struct *tsk);
+int bc_task_move(int pid, struct beancounter *bc, int whole);
+#else
+static inline int __must_check copy_beancounter(struct task_struct *tsk,
```

```
+  struct task_struct *parent)
+{
+ return 0;
+}
+
+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.bcctx 2006-11-09 11:29:12.000000000 +0300
+++ ./include/linux/sched.h 2006-11-09 11:32:24.000000000 +0300
@@ -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>
@@ -1061,6 +1063,9 @@ 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)
--- ./kernel/bc/beancounter.c.bcctx 2006-11-09 11:30:04.000000000 +0300
+++ ./kernel/bc/beancounter.c 2006-11-09 11:32:24.000000000 +0300
@@ -235,6 +235,8 @@ void __init bc_init_early(void)

  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);
 }

 int __init bc_init_late(void)
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/misc.c 2006-11-09 11:32:24.000000000 +0300
@@ -0,0 +1,100 @@
```

```
+/*
+ * kernel/bc/misc.c
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#include <linux/sched.h>
+#include <linux/stop_machine.h>
+#include <linux/module.h>
+
+#include <bc/beancounter.h>
+#include <bc/task.h>
+#include <bc/misc.h>
+
+int copy_beancounter(struct task_struct *tsk, struct task_struct *parent)
+{
+ struct beancounter *bc;
+
+ bc = parent->exec_bc;
+ tsk->exec_bc = bc_get(bc);
+ return 0;
+}
+
+void free_beancounter(struct task_struct *tsk)
+{
+ struct beancounter *bc;
+
+ bc = tsk->exec_bc;
+ bc_put(bc);
+}
+
+struct set_bcid_data {
+ struct beancounter *bc;
+ struct mm_struct *mm;
+ struct task_struct *tsk;
+ int whole;
+};
+
+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;
+
+ if (!d->whole) {
```

```
+  p = d->tsk;
+  bc_put(p->exec_bc);
+  p->exec_bc = bc_get(d->bc);
+  return 0;
+ }
+
+ 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(int pid, struct beancounter *bc, int whole)
+{
+ int err;
+ struct set_bcid_data data;
+ struct task_struct *tsk;
+ struct mm_struct *mm;
+
+ read_lock(&tasklist_lock);
+ tsk = find_task_by_pid(pid);
+ if (tsk)
+  get_task_struct(tsk);
+ read_unlock(&tasklist_lock);
+ if (tsk == NULL)
+  return -ESRCH;
+
+ mm = get_task_mm(tsk);
+ if (mm == NULL)
+  return -EINVAL;
+
+ data.bc = bc;
+ data.mm = mm;
+ data.tsk = tsk;
+ data.whole = whole;
+
+ down_write(&mm->mmap_sem);
+ err = stop_machine_run(do_set_bcid, &data, NR_CPUS);
+ up_write(&mm->mmap_sem);
+
+ mmput(mm);
```

```
+ put_task_struct(tsk);
+ return err;
+}
+EXPORT_SYMBOL(bc_task_move);
--- ./kernel/fork.c.bcctx 2006-11-09 11:29:12.000000000 +0300
+++ ./kernel/fork.c 2006-11-09 11:32:24.000000000 +0300
@@ -49,6 +49,8 @@
 #include <linux/taskstats_kern.h>
 #include <linux/random.h>

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

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

 void __put_task_struct(struct task_struct *tsk)
@@ -985,6 +993,10 @@ static struct task_struct *copy_process(

  rt_mutex_init_task(p);

+ retval = copy_beancounter(p, current);
+ if (retval < 0)
+  goto bad_fork_bc;
+
 #ifdef CONFIG_TRACE_IRQFLAGS
  DEBUG_LOCKS_WARN_ON(!p->hardirqs_enabled);
  DEBUG_LOCKS_WARN_ON(!p->softirqs_enabled);
@@ -1298,7 +1310,9 @@ bad_fork_cleanup_count:
  atomic_dec(&p->user->processes);
```

```
  free_uid(p->user);
 bad_fork_free:
- free_task(p);
+ free_beancounter(p);
+bad_fork_bc:
+ __free_task(p);
 fork_out:
  return ERR_PTR(retval);
 }
--- ./kernel/irq/handle.c.bcctx 2006-11-09 11:29:12.000000000 +0300
+++ ./kernel/irq/handle.c 2006-11-09 11:32:24.000000000 +0300
@@ -16,6 +16,8 @@
 #include <linux/interrupt.h>
 #include <linux/kernel_stat.h>

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

 /**
@@ -169,6 +171,7 @@ 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)) {
@@ -225,6 +228,8 @@ 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;

@@ -239,6 +244,7 @@ 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.bcctx 2006-11-09 11:29:12.000000000 +0300
+++ ./kernel/softirq.c 2006-11-09 11:32:24.000000000 +0300
@@ -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 5/13] BC: configfs interface
Posted by dev on Thu, 09 Nov 2006 16:52:24 GMT

View Forum Message <> Reply to Message

configfs interface to beancounters.
It may be done as module in case configfs itself is a module.

Usage example:

Prepare configfs interface
~~~~~~~~~~~~~~~~~~~~~~~~~~
```
# mount -t configfs none /cfg
# ls /cfg
  beancounters
# ls /cfg/beancounters/
```
~~~~~~~~~~~~~~~~~~~~~~~~~~

```
  0  description
# cat /cfg/beancounters/description
  Beancounters controll resource usage of task groups
```

Look at beancounters subsystem structure
~~~~~~

```
# ls /cfg/beancounters/0/
    id  kmemsize  numfiles  numtasks  physpages  privvmpages  tasks  threads
# ls /cfg/beancounters/0/numfiles
    barrier  failcnt  held  limit  maxheld  minheld
# cat /cfg/beancounters/0/numfiles/*
    2147483647
    0
    163
    2147483647
    345
    0
```

Create a new beancounter and move task into it
~~~~~~

```
# mkdir /cfg/beancounters/1
# echo -n $$ > /cfg/beancounters/1/tasks
# cat /proc/$$/status | grep Bcid
Bcid:   1
```

now you can see bc 1 usages etc.

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

```
 configfs.c |  391  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+
 1 files changed, 391 insertions(+)

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/configfs.c 2006-11-03 17:13:25.000000000 +0300
@@ -0,0 +1,391 @@
+/*
+ * kernel/bc/configfs.c
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ * Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
```

```
+#include <linux/configfs.h>
+#include <linux/module.h>
+
+#include <bc/beancounter.h>
+#include <bc/task.h>
+
+/*
+ * Declarations
+ */
+
+enum bc_attr_indices {
+ BC_ATTR_ID,
+ BC_ATTR_TASKS,
+ BC_ATTR_THREADS,
+ BC_RES_HELD,
+ BC_RES_MAXHELD,
+ BC_RES_MINHELD,
+ BC_RES_BARRIER,
+ BC_RES_LIMIT,
+ BC_RES_FAILCNT,
+};
+
+struct bc_configfs_attribute {
+ int idx;
+ struct configfs_attribute attr;
+};
+
+#define attr_to_bc(attr) container_of(attr, \
+  struct bc_configfs_attribute, attr)
+
+struct bc_resource_group {
+ int res;
+ struct config_group group;
+};
+
+#define item_to_res(i) container_of(i, \
+  struct bc_resource_group, group.cg_item)
+
+struct bc_group {
+ struct beancounter *bc;
+ struct config_group group;
+};
+
+#define item_to_bc(i) container_of(i, \
+  struct bc_group, group.cg_item)
+
+#define declare_bc_attr(i, name, mode)    \
+ static struct bc_configfs_attribute bc_attr_##name = { \
```

```
+  .idx = i,     \
+  .attr = {     \
+   .ca_owner = THIS_MODULE,  \
+   .ca_name = #name,   \
+   .ca_mode = mode,   \
+  },      \
+ }
+
+declare_bc_attr(BC_ATTR_ID, id, S_IRUGO);
+declare_bc_attr(BC_ATTR_TASKS, tasks, S_IWUSR);
+declare_bc_attr(BC_ATTR_THREADS, threads, S_IWUSR);
+declare_bc_attr(BC_RES_HELD, held, S_IRUGO);
+declare_bc_attr(BC_RES_BARRIER, barrier, S_IRUGO | S_IWUSR);
+declare_bc_attr(BC_RES_LIMIT, limit, S_IRUGO | S_IWUSR);
+declare_bc_attr(BC_RES_MAXHELD, maxheld, S_IRUGO);
+declare_bc_attr(BC_RES_MINHELD, minheld, S_IRUGO);
+declare_bc_attr(BC_RES_FAILCNT, failcnt, S_IRUGO);
+
+static struct configfs_attribute *bc_attributes[] = {
+ &bc_attr_id.attr,
+ &bc_attr_tasks.attr,
+ &bc_attr_threads.attr,
+ NULL,
+};
+
+static struct configfs_attribute *resource_attributes[] = {
+ &bc_attr_held.attr,
+ &bc_attr_barrier.attr,
+ &bc_attr_limit.attr,
+ &bc_attr_maxheld.attr,
+ &bc_attr_minheld.attr,
+ &bc_attr_failcnt.attr,
+ NULL,
+};
+
+static ssize_t bc_show(struct config_item *item,
+  struct configfs_attribute *attr, char *page)
+{
+ struct beancounter *bc;
+
+ bc = item_to_bc(item)->bc;
+
+ switch (attr_to_bc(attr)->idx) {
+ case BC_ATTR_ID:
+  return sprintf(page, "%d\n", bc->bc_id);
+ }
+
+ return 0;
```

```
+}
+
+static ssize_t bc_store(struct config_item *item,
+  struct configfs_attribute *attr,
+  const char *page, size_t size)
+{
+ struct beancounter *bc;
+ char *end;
+ int ret;
+
+ ret = simple_strtoul(page, &end, 10);
+ if (*end != '\0')
+  return -EINVAL;
+
+ bc = item_to_bc(item)->bc;
+
+ switch (attr_to_bc(attr)->idx) {
+ case BC_ATTR_TASKS:
+  ret = bc_task_move(ret, bc, 1);
+  if (ret == 0)
+   ret = size;
+  break;
+ case BC_ATTR_THREADS:
+  ret = bc_task_move(ret, bc, 0);
+  if (ret == 0)
+   ret = size;
+  break;
+ };
+
+ return ret;
+}
+
+static ssize_t resource_show(struct config_item *item,
+  struct configfs_attribute *attr, char *page)
+{
+ struct beancounter *bc;
+ struct bc_resource_group *grp;
+
+ bc = item_to_bc(item->ci_parent)->bc;
+ grp = item_to_res(item);
+
+ switch (attr_to_bc(attr)->idx) {
+ case BC_RES_HELD:
+  return sprintf(page, "%lu\n", bc->bc_parms[grp->res].held);
+ case BC_RES_BARRIER:
+  return sprintf(page, "%lu\n", bc->bc_parms[grp->res].barrier);
+ case BC_RES_LIMIT:
+  return sprintf(page, "%lu\n", bc->bc_parms[grp->res].limit);
```

```
+ case BC_RES_MAXHELD:
+  return sprintf(page, "%lu\n", bc->bc_parms[grp->res].maxheld);
+ case BC_RES_MINHELD:
+  return sprintf(page, "%lu\n", bc->bc_parms[grp->res].minheld);
+ case BC_RES_FAILCNT:
+  return sprintf(page, "%lu\n", bc->bc_parms[grp->res].failcnt);
+ }
+
+ return 0;
+}
+
+static ssize_t resource_store(struct config_item *item,
+  struct configfs_attribute *attr,
+  const char *page, size_t size)
+{
+ struct beancounter *bc;
+ struct bc_resource_group *grp;
+ unsigned long val;
+ char *end;
+ int ret;
+
+ bc = item_to_bc(item->ci_parent)->bc;
+ grp = item_to_res(item);
+
+ ret = -EINVAL;
+ val = simple_strtoul(page, &end, 10);
+ if (*end != '\0')
+  goto out;
+
+ switch (attr_to_bc(attr)->idx) {
+ case BC_RES_BARRIER:
+  ret = bc_change_param(bc, grp->res,
+    val, bc->bc_parms[grp->res].limit);
+  if (ret == 0)
+   ret = size;
+  break;
+ case BC_RES_LIMIT:
+  ret = bc_change_param(bc, grp->res,
+    bc->bc_parms[grp->res].barrier, val);
+  if (ret == 0)
+   ret = size;
+  break;
+ }
+out:
+ return ret;
+}
+
+static void bc_release(struct config_item *item)
```

```
+{
+ kfree(to_config_group(item)->default_groups);
+ bc_put(item_to_bc(item)->bc);
+}
+
+static void resource_release(struct config_item *item)
+{
+ kfree(to_config_group(item));
+}
+
+static struct configfs_item_operations bc_item_ops = {
+ .show_attribute = bc_show,
+ .store_attribute = bc_store,
+ .release = bc_release,
+};
+
+static struct config_item_type bc_item_type = {
+ .ct_item_ops = &bc_item_ops,
+ .ct_attrs = bc_attributes,
+ .ct_owner = THIS_MODULE,
+};
+
+static struct configfs_item_operations resource_item_ops = {
+ .show_attribute = resource_show,
+ .store_attribute = resource_store,
+ .release = resource_release,
+};
+
+static struct config_item_type resource_item_type = {
+ .ct_item_ops = &resource_item_ops,
+ .ct_attrs = resource_attributes,
+ .ct_owner = THIS_MODULE,
+};
+
+static int bc_init_default_groups(struct config_group *group)
+{
+ int i;
+ struct bc_resource_group *def_group;
+
+ group->default_groups = kmalloc((BC_RESOURCES + 1) *
+   sizeof(struct config_group *), GFP_KERNEL);
+ if (group->default_groups == NULL)
+  goto out;
+
+ for (i = 0; i < BC_RESOURCES; i++) {
+  def_group = kzalloc(sizeof(struct bc_resource_group),
+    GFP_KERNEL);
+  if (def_group == NULL)
```

```
+   goto out_free;
+
+  def_group->res = i;
+  config_group_init_type_name(&def_group->group,
+    bc_resources[i]->bcr_name, &resource_item_type);
+  group->default_groups[i] = &def_group->group;
+ }
+
+ group->default_groups[i] = NULL;
+ return 0;
+
+out_free:
+ for (i--; i >= 0; i--)
+  config_group_put(group->default_groups[i]);
+
+ kfree(group->default_groups);
+out:
+ return -ENOMEM;
+}
+
+static struct config_group *bc_new_group(struct beancounter *bc,
+  const char *name)
+{
+ struct bc_group *bc_group;
+
+ bc_group = kzalloc(sizeof(struct bc_group), GFP_KERNEL);
+ if (bc_group == NULL)
+  goto out;
+
+ config_group_init_type_name(&bc_group->group, name, &bc_item_type);
+ if (bc_init_default_groups(&bc_group->group))
+  goto out_free;
+
+ bc_group->bc = bc;
+ return &bc_group->group;
+
+out_free:
+ config_group_put(&bc_group->group);
+out:
+ return NULL;
+}
+
+static struct config_group *bc_make_group(struct config_group *group,
+  const char *name)
+{
+ int id;
+ char *end;
+ struct beancounter *bc;
```

```
+ struct config_group *grp;
+
+ id = simple_strtoul(name, &end, 10);
+ if (*end != '\0')
+  goto out;
+
+ bc = bc_findcreate(id, BC_ALLOC);
+ if (bc == NULL)
+  goto out;
+
+ grp = bc_new_group(bc, name);
+ if (grp == NULL)
+  goto out_put;
+
+ return grp;
+
+out_put:
+ bc_put(bc);
+out:
+ return NULL;
+}
+
+/* subsystem description */
+
+static struct configfs_group_operations bc_group_group_ops = {
+ .make_group = bc_make_group,
+};
+
+static ssize_t bc_group_attr_show(struct config_item *item,
+  struct configfs_attribute *attr,
+  char *page)
+{
+ return sprintf(page, "Beancounters "
+   "controll resource usage of task groups\n");
+}
+
+static struct configfs_item_operations bc_group_item_ops = {
+ .show_attribute = bc_group_attr_show,
+};
+
+static struct configfs_attribute bc_attr_description = {
+ .ca_owner = THIS_MODULE,
+ .ca_name = "description",
+ .ca_mode = S_IRUGO,
+};
+
+static struct configfs_attribute *bc_group_attributes[] = {
+ &bc_attr_description,
```

```
+ NULL,
+};
+
+static struct config_item_type bc_group_type = {
+ .ct_item_ops = &bc_group_item_ops,
+ .ct_group_ops = &bc_group_group_ops,
+ .ct_attrs = bc_group_attributes,
+ .ct_owner = THIS_MODULE,
+};
+
+struct configfs_subsystem bc_subsystem = {
+ .su_group = {
+  .cg_item = {
+   .ci_namebuf = "beancounters",
+   .ci_type = &bc_group_type,
+  },
+ },
+};
+
+int __init bc_init_configfs(void)
+{
+ static struct config_group *subsys_default_groups[2];
+ struct config_group *bc_group;
+
+ bc_group = bc_new_group(&init_bc, "0");
+ if (bc_group == NULL)
+  return -ENOMEM;
+
+ subsys_default_groups[0] = bc_group;
+ subsys_default_groups[1] = NULL;
+ bc_subsystem.su_group.default_groups = subsys_default_groups;
+
+ config_group_init(&bc_subsystem.su_group);
+ init_MUTEX(&bc_subsystem.su_sem);
+
+ return configfs_register_subsystem(&bc_subsystem);
+}
+
+static void __exit bc_fini_configfs(void)
+{
+ configfs_unregister_subsystem(&bc_subsystem);
+}
+
+module_init(bc_init_configfs);
+module_exit(bc_fini_configfs);
+MODULE_LICENSE("GPL");
```

Subject: [PATCH 6/13] BC: kmemsize accounting (core)
Posted by dev on Thu, 09 Nov 2006 16:53:38 GMT
View Forum Message <> Reply to Message

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

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

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

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

```
 include/bc/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/slab.c               |   91 ++++++++++++++++++++++++++++++++--------
 mm/vmalloc.c            |    6 ++
 9 files changed, 253 insertions(+), 21 deletions(-)

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/kmem.h 2006-11-03 15:48:26.000000000 +0300
@@ -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 2006-11-03 15:35:28.000000000 +0300
+++ ./include/linux/gfp.h 2006-11-03 15:48:26.000000000 +0300
@@ -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 2006-11-03 15:35:28.000000000 +0300
+++ ./include/linux/mm.h 2006-11-03 15:48:26.000000000 +0300
@@ -219,6 +222,7 @@ 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 2006-11-03 15:35:28.000000000 +0300
+++ ./include/linux/mm_types.h 2006-11-03 15:48:26.000000000 +0300
@@ -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 2006-11-03 15:35:28.000000000 +0300
+++ ./include/linux/slab.h 2006-11-03 15:48:26.000000000 +0300
@@ -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 2006-11-03 15:35:28.000000000 +0300
+++ ./include/linux/vmalloc.h 2006-11-03 15:48:26.000000000 +0300
@@ -37,6 +37,7 @@ 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 2006-11-03 15:48:26.000000000 +0300
@@ -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, int res)
+{
+ bc_init_resource(&bc->bc_parms[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/slab.c.bc 2006-11-03 15:35:28.000000000 +0300
+++ ./mm/slab.c 2006-11-03 15:48:26.000000000 +0300
@@ -109,6 +109,8 @@
 #include <linux/rtmutex.h>
```

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

+#include <bc/kmem.h>
+
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>
 #include <asm/page.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

@@ -796,9 +800,33 @@ 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);
 }

 /*
@@ -843,20 +871,21 @@ 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;
@@ -1989,7 +2021,8 @@ 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;
@@ -2291,8 +2324,8 @@ 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
@@ -2303,11 +2336,9 @@ 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. */
@@ -2548,6 +2579,30 @@ 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
@@ -2568,7 +2623,8 @@ 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 {
@@ -2579,14 +2635,14 @@ 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)
 {
@@ -2766,7 +2822,7 @@ 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;

--- ./mm/vmalloc.c.bc 2006-11-03 15:35:28.000000000 +0300
+++ ./mm/vmalloc.c 2006-11-03 15:48:26.000000000 +0300
@@ -517,6 +517,12 @@ 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 7/13] BC: kmemsize accounting (hooks)
Posted by dev on Thu, 09 Nov 2006 16:54:37 GMT
View Forum Message <> Reply to Message

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

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

```
 arch/i386/kernel/ldt.c          |    4 ++--
 arch/i386/mm/init.c             |    4 ++--
 arch/i386/mm/pgtable.c          |    6 ++++--
 drivers/char/tty_io.c           |   10 +++++-----
 fs/file.c                       |    6 +++---
 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/mempool.c                    |    2 ++
 mm/page_alloc.c                 |   11 +++++++++++
 mm/slab.c                       |   30 ++++++++++++++++++++++++++++---
 22 files changed, 115 insertions(+), 56 deletions(-)
```

```
--- ./arch/i386/kernel/ldt.c.bckmem 2006-11-09 11:29:09.000000000 +0300
+++ ./arch/i386/kernel/ldt.c 2006-11-09 11:33:27.000000000 +0300
@@ -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.bckmem 2006-11-09 11:18:45.000000000 +0300
+++ ./arch/i386/mm/init.c 2006-11-09 11:33:27.000000000 +0300
@@ -708,7 +708,7 @@ 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)
@@ -717,7 +717,7 @@ 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.bckmem 2006-11-09 11:18:45.000000000 +0300
+++ ./arch/i386/mm/pgtable.c 2006-11-09 11:33:27.000000000 +0300
@@ -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.bckmem 2006-11-09 11:29:10.000000000 +0300
+++ ./drivers/char/tty_io.c 2006-11-09 11:33:27.000000000 +0300
@@ -167,7 +167,7 @@ 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 *);
@@ -1932,7 +1932,7 @@ static int init_dev(struct tty_driver *d

  if (!*tp_loc) {
   tp = (struct ktermios *) kmalloc(sizeof(struct ktermios),
-      GFP_KERNEL);
+      GFP_KERNEL_BC);
   if (!tp)
    goto free_mem_out;
   *tp = driver->init_termios;
@@ -1940,7 +1940,7 @@ static int init_dev(struct tty_driver *d

  if (!*ltp_loc) {
   ltp = (struct ktermios *) kmalloc(sizeof(struct ktermios),
-       GFP_KERNEL);
+       GFP_KERNEL_BC);
   if (!ltp)
    goto free_mem_out;
   memset(ltp, 0, sizeof(struct ktermios));
@@ -1965,7 +1965,7 @@ static int init_dev(struct tty_driver *d

   if (!*o_tp_loc) {
    o_tp = (struct ktermios *)
-    kmalloc(sizeof(struct ktermios), GFP_KERNEL);
+    kmalloc(sizeof(struct ktermios), GFP_KERNEL_BC);
    if (!o_tp)
     goto free_mem_out;
    *o_tp = driver->other->init_termios;
@@ -1973,7 +1973,7 @@ static int init_dev(struct tty_driver *d

   if (!*o_ltp_loc) {
    o_ltp = (struct ktermios *)
-    kmalloc(sizeof(struct ktermios), GFP_KERNEL);
+    kmalloc(sizeof(struct ktermios), GFP_KERNEL_BC);
    if (!o_ltp)
```

```
     goto free_mem_out;
    memset(o_ltp, 0, sizeof(struct ktermios));
--- ./fs/file.c.bckmem 2006-11-09 11:29:11.000000000 +0300
+++ ./fs/file.c 2006-11-09 11:33:27.000000000 +0300
@@ -35,9 +35,9 @@ static DEFINE_PER_CPU(struct fdtable_def
 static inline void * alloc_fdmem(unsigned int size)
 {
  if (size <= PAGE_SIZE)
-  return kmalloc(size, GFP_KERNEL);
+  return kmalloc(size, GFP_KERNEL_BC);
  else
-  return vmalloc(size);
+  return vmalloc_bc(size);
 }

 static inline void free_fdarr(struct fdtable *fdt)
@@ -148,7 +148,7 @@ static struct fdtable * alloc_fdtable(un
  if (nr > NR_OPEN)
   nr = NR_OPEN;

- fdt = kmalloc(sizeof(struct fdtable), GFP_KERNEL);
+ fdt = kmalloc(sizeof(struct fdtable), GFP_KERNEL_BC);
  if (!fdt)
   goto out;
  fdt->max_fds = nr;
--- ./fs/locks.c.bckmem 2006-11-09 11:29:11.000000000 +0300
+++ ./fs/locks.c 2006-11-09 11:33:27.000000000 +0300
@@ -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.bckmem 2006-11-09 11:29:11.000000000 +0300
+++ ./fs/namespace.c 2006-11-09 11:33:27.000000000 +0300
@@ -1813,7 +1813,8 @@ void __init mnt_init(unsigned long mempa
  init_rwsem(&namespace_sem);

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

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

```
--- ./fs/select.c.bckmem 2006-11-09 11:29:12.000000000 +0300
+++ ./fs/select.c 2006-11-09 11:33:27.000000000 +0300
@@ -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.bckmem 2006-11-09 11:29:12.000000000 +0300
+++ ./include/asm-i386/thread_info.h 2006-11-09 11:33:27.000000000 +0300
@@ -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.bckmem 2006-09-20 14:46:38.000000000 +0400
+++ ./include/asm-ia64/pgalloc.h 2006-11-09 11:33:27.000000000 +0300
@@ -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.bckmem 2006-09-20 14:46:40.000000000 +0400
+++ ./include/asm-x86_64/pgalloc.h 2006-11-09 11:33:27.000000000 +0300
@@ -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.bckmem 2006-11-09 11:19:09.000000000 +0300
+++ ./include/asm-x86_64/thread_info.h 2006-11-09 11:33:27.000000000 +0300
@@ -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.bckmem 2006-11-09 11:19:09.000000000 +0300
+++ ./ipc/msgutil.c 2006-11-09 11:33:27.000000000 +0300
@@ -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.bckmem 2006-11-09 11:29:12.000000000 +0300
+++ ./ipc/sem.c 2006-11-09 11:33:27.000000000 +0300
@@ -1009,7 +1009,7 @@ 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);
@@ -1072,7 +1072,8 @@ 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);
@@ -1133,7 +1134,7 @@ 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.bckmem 2006-11-09 11:22:58.000000000 +0300
+++ ./ipc/util.c 2006-11-09 11:33:27.000000000 +0300
@@ -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.bckmem 2006-11-09 11:32:24.000000000 +0300
+++ ./kernel/fork.c 2006-11-09 11:33:27.000000000 +0300
@@ -143,7 +143,7 @@ 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

  /*
@@ -1441,23 +1441,24 @@ 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.bckmem 2006-11-09 11:29:12.000000000 +0300
+++ ./kernel/posix-timers.c 2006-11-09 11:33:27.000000000 +0300
@@ -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.bckmem 2006-11-09 11:29:12.000000000 +0300
+++ ./kernel/signal.c 2006-11-09 11:33:27.000000000 +0300
@@ -2764,5 +2764,5 @@ 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.bckmem 2006-11-09 11:29:56.000000000 +0300
+++ ./kernel/user.c 2006-11-09 11:33:27.000000000 +0300
@@ -205,7 +205,7 @@ 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/mempool.c.bckmem 2006-09-20 14:46:41.000000000 +0400
+++ ./mm/mempool.c 2006-11-09 11:33:27.000000000 +0300
@@ -119,6 +119,7 @@ int mempool_resize(mempool_t *pool, int
  unsigned long flags;

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

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

  gfp_temp = gfp_mask & ~(__GFP_WAIT|__GFP_IO);

--- ./mm/page_alloc.c.bckmem 2006-11-09 11:29:12.000000000 +0300
+++ ./mm/page_alloc.c 2006-11-09 11:33:27.000000000 +0300
@@ -41,6 +41,8 @@
 #include <linux/pfn.h>
 #include <linux/backing-dev.h>

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

```
 #include "internal.h"
@@ -513,6 +515,8 @@ static void __free_pages_ok(struct page
  if (reserved)
   return;

+ bc_page_uncharge(page, order);
+
  if (!PageHighMem(page))
   debug_check_no_locks_freed(page_address(page),PAGE_SIZE<<order);
  arch_free_page(page, order);
@@ -800,6 +804,8 @@ static void fastcall free_hot_cold_page(
  if (free_pages_check(page))
   return;

+ bc_page_uncharge(page, 0);
+
  if (!PageHighMem(page))
   debug_check_no_locks_freed(page_address(page), PAGE_SIZE);
  arch_free_page(page, 0);
@@ -1337,6 +1343,11 @@ nopage:
   show_mem();
  }
 got_pg:
+ if ((gfp_mask & __GFP_BC) &&
+    bc_page_charge(page, order, gfp_mask)) {
+  __free_pages(page, order);
+  page = NULL;
+ }
 #ifdef CONFIG_PAGE_OWNER
  if (page)
   set_page_owner(page, order, gfp_mask);
--- ./mm/slab.c.bckmem 2006-11-09 11:33:17.000000000 +0300
+++ ./mm/slab.c 2006-11-09 11:33:27.000000000 +0300
@@ -1483,7 +1483,8 @@ 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) {
@@ -1491,7 +1492,8 @@ 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);
  }

@@ -1509,7 +1511,8 @@ 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);
    }
    if (CONFIG_ZONE_DMA_FLAG)
@@ -3168,6 +3171,19 @@ 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)
 {
@@ -3193,6 +3209,12 @@ 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;
 }
```

```
@@ -3419,6 +3441,8 @@ 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))
```

Subject: [PATCH 8/13] BC: privvmpages accounting (core)
Posted by dev on Thu, 09 Nov 2006 16:56:01 GMT
View Forum Message <> Reply to Message

This patch instroduces new resource - BC_PRIVVMPAGES.
It is an upper estimation of currently used physical memory.

There are different approaches to user pages control:
a) account all the mappings on mmap/brk and reject as
   soon as the sum of VMA's lengths reaches the barrier.

   This approach is very bad as applications always map
   more than they really use, very often MUCH more.

b) account only the really used memory and reject as
   soon as RSS reaches the limit.

   This approach is not good either as user space pages are
   allocated in page fault handler and the only way to reject
   allocation is to kill the task.

   Comparing to previous scenarion this is much worse as
   application won't even be able to terminate gracefully.

c) account a part of memory on mmap/brk and reject there,
   and account the rest of the memory in page fault handlers
   without any rejects.
   This type of accounting is used in UBC.

d) account physical memory and behave like a standalone
   kernel - reclaim user memory when run out of it.

   This type of memory control is to be introduced later
   as an addition to c). UBC provides all the needed
   statistics for this (physical memory, swap pages etc.)

Privvmpages accounting is described in details in

http://wiki.openvz.org/User_pages_accounting

A note about sys_mprotect: as it can change mapping state from
bc_vm_private to !bc_vm_private and vice-versa appropriate amount of
pages is (un)charged in mprotect_fixup.

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

---

 include/bc/vmpages.h | 90 ++++++++++++++++++++++++++++++++++
 include/linux/mm.h | 3 +
 include/linux/sched.h | 3 +
 kernel/bc/beancounter.c | 1
 kernel/bc/vmpages.c | 138 ++
 5 files changed, 235 insertions(+)

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/vmpages.h 2006-11-03 17:49:13.000000000 +0300
@@ -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.bcvmpcore 2006-11-03 17:48:37.000000000 +0300
+++ ./include/linux/mm.h 2006-11-03 17:49:13.000000000 +0300
@@ -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.bcvmpcore 2006-11-03 17:47:38.000000000 +0300
+++ ./include/linux/sched.h 2006-11-03 17:49:13.000000000 +0300
@@ -374,6 +374,9 @@ 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 {
--- ./kernel/bc/beancounter.c.bcvmpcore 2006-11-03 17:47:38.000000000 +0300
+++ ./kernel/bc/beancounter.c 2006-11-03 17:49:36.000000000 +0300
@@ -237,6 +237,7 @@ void __init bc_init_early(void)
  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)
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/vmpages.c 2006-11-03 17:49:13.000000000 +0300
@@ -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, int res)
+{
+ bc_init_resource(&bc->bc_parms[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 9/13] BC: privvmpages accounting (hooks)
Posted by dev on Thu, 09 Nov 2006 16:57:24 GMT
View Forum Message <> Reply to Message

BC privvmpages accounting hooks in generic code.

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

```
---

 fs/binfmt_elf.c            |    5 +--
 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/mlock.c                 |   16 +++++++++
 mm/mmap.c                  |   74 ++++++++++++++++++++++++++++++++++++++++-------
 mm/mprotect.c              |   18 ++++++++++
 mm/mremap.c                |   22 ++++++++++---
 mm/shmem.c                 |   23 +++++++++++++
 12 files changed, 153 insertions(+), 21 deletions(-)

--- ./fs/binfmt_elf.c.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./fs/binfmt_elf.c 2006-11-03 17:50:04.000000000 +0300
@@ -363,7 +363,7 @@ 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;
@@ -849,7 +849,8 @@ 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) {
--- ./include/asm-alpha/mman.h.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./include/asm-alpha/mman.h 2006-11-03 17:50:04.000000000 +0300
@@ -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.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./include/asm-generic/mman.h 2006-11-03 17:50:04.000000000 +0300
@@ -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.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./include/asm-mips/mman.h 2006-11-03 17:50:04.000000000 +0300
@@ -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.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./include/asm-parisc/mman.h 2006-11-03 17:50:04.000000000 +0300
@@ -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.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./include/asm-xtensa/mman.h 2006-11-03 17:50:04.000000000 +0300
@@ -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.bcvmpagesk 2006-11-03 17:48:46.000000000 +0300
```

```
+++ ./kernel/fork.c 2006-11-03 17:50:04.000000000 +0300
@@ -50,6 +50,7 @@ static inline int dup_mmap(struct mm_str
 #include <linux/random.h>

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

 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
@@ -257,6 +257,9 @@ 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) {
@@ -299,6 +302,10 @@ 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:
@@ -349,6 +356,7 @@ 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;
  }
@@ -379,6 +387,7 @@ 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);
@@ -526,6 +535,7 @@ 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/mlock.c.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./mm/mlock.c 2006-11-03 17:50:04.000000000 +0300
@@ -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.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./mm/mmap.c 2006-11-03 17:50:04.000000000 +0300
@@ -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;
@@ -1054,6 +1065,10 @@ 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
@@ -1082,6 +1097,7 @@ 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;
@@ -1138,6 +1154,7 @@ munmap_back:
    fput(file);
   }
   mpol_free(vma_policy(vma));
+  vma_release_bc(vma);
   kmem_cache_free(vm_area_cachep, vma);
  }
 out:
@@ -1167,6 +1184,8 @@ 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;
@@ -1496,12 +1515,16 @@ 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;
@@ -1509,6 +1532,11 @@ 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)
@@ -1742,6 +1770,7 @@ 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;
@@ -1752,6 +1781,7 @@ 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);
  }
@@ -1864,7 +1894,8 @@ 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;
@@ -1924,7 +1955,10 @@ 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,
@@ -1935,10 +1969,8 @@ 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;
@@ -1947,6 +1979,7 @@ 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;
@@ -1955,8 +1988,19 @@ 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. */
@@ -2018,6 +2062,12 @@ 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;
 }
@@ -2057,8 +2107,10 @@ 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.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./mm/mprotect.c 2006-11-03 17:50:04.000000000 +0300
@@ -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.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./mm/mremap.c 2006-11-03 17:50:04.000000000 +0300
@@ -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/shmem.c.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./mm/shmem.c 2006-11-03 17:50:04.000000000 +0300
@@ -50,6 +50,8 @@
 #include <linux/highmem.h>
 #include <linux/backing-dev.h>

+#include <bc/vmpages.h>
+
 #include <asm/uaccess.h>
 #include <asm/div64.h>
 #include <asm/pgtable.h>
@@ -372,7 +374,8 @@ 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);
@@ -2525,6 +2528,24 @@ int shmem_zero_setup(struct vm_area_stru

  if (vma->vm_file)
   fput(vma->vm_file);
+ else {
+  /*
+   * vma obains file which was not on it when bc_memory_charge()
+   * was called in do_mmap_pgoff, so we must check whether or
+   * not to recharge private memory
+   */
+  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: [PATCH 10/13] BC: physpages accounting (core)
Posted by dev on Thu, 09 Nov 2006 16:59:40 GMT
View Forum Message <> Reply to Message

This is the core of vmrss accounting.

The main introduced object is page_beancounter.
It ties together page and BCs which use the page.
page_beancounter also allows quick per-container pages reclamation.
and helps correctly account fractions of memory shared
between BCs (http://wiki.openvz.org/RSS_fractions_accounting)

Page charge/uncharge is performed on first map/last unmap
and is based on page->mapcount calculations.

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

```
 include/bc/beancounter.h |    2
 include/bc/rsspages.h    |   46 ++++++++
 include/linux/mm.h       |    5
 include/linux/mm_types.h |    5
 kernel/bc/beancounter.c  |    2
 kernel/bc/rsspages.c     |  267 +++++++++++++++++++++++++++++++++++++++++++++++
 6 files changed, 326 insertions(+), 1 deletion(-)

--- ./include/bc/beancounter.h.ve9 2006-11-07 12:03:41.000000000 +0300
+++ ./include/bc/beancounter.h 2006-11-07 12:03:47.000000000 +0300
@@ -67,6 +67,8 @@ struct beancounter {
  bcid_t   bc_id;
  struct hlist_node bc_hash;

+ spinlock_t  bc_page_lock;
+ struct list_head bc_page_list;
  struct bc_resource_parm bc_parms[BC_RESOURCES];
 };

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
```

```
+++ ./include/bc/rsspages.h 2006-11-07 12:03:57.000000000 +0300
@@ -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_beancounter *pb);
+
+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_beancounter *pb)
+{
+}
+#endif
+#endif
```

```
--- ./include/linux/mm.h.ve9 2006-11-07 12:03:41.000000000 +0300
+++ ./include/linux/mm.h 2006-11-07 12:03:47.000000000 +0300
@@ -223,6 +223,11 @@ struct mmu_gather;
 struct inode;

 #define page_bc(page)   ((page)->bc)
+#ifdef CONFIG_BEANCOUNTERS
+#define page_pb(page)   ((page)->pb)
+#else
+#define page_pb(page)   (NULL)
+#endif
 #define page_private(page)  ((page)->private)
 #define set_page_private(page, v) ((page)->private = (v))

--- ./include/linux/mm_types.h.ve9 2006-11-07 12:03:41.000000000 +0300
+++ ./include/linux/mm_types.h 2006-11-07 12:03:47.000000000 +0300
@@ -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;
--- ./kernel/bc/beancounter.c.ve9 2006-11-07 12:03:41.000000000 +0300
+++ ./kernel/bc/beancounter.c 2006-11-07 12:03:47.000000000 +0300
@@ -227,6 +227,8 @@ 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;
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/rsspages.c 2006-11-07 12:03:47.000000000 +0300
@@ -0,0 +1,267 @@
+/*
+ * 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
+ * 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;
+ 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_beancounter *pb)
+{
+ struct page *page;
+ struct beancounter *bc;
+
+ if (pb == NULL)
+  return;
+
+ page = pb->page;
+ bc = pb->bc;
+
+ cmpxchg(&page_pb(page), pb, NULL);
+
+ spin_lock(&bc->bc_page_lock);
+ list_del(&pb->list);
+ spin_unlock(&bc->bc_page_lock);
+
+ bc_uncharge(bc, BC_PHYSPAGES, 1);
+ kfree(pb);
+
+ bc_put(bc);
+}
+
+/*
+ * 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, int res)
+{
+ spin_lock_init(&bc->bc_page_lock);
+ INIT_LIST_HEAD(&bc->bc_page_list);
+
+ bc_init_resource(&bc->bc_parms[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);
```

Subject: [PATCH 11/13] BC: physpages accounting (hooks)
Posted by dev on Thu, 09 Nov 2006 17:01:28 GMT
View Forum Message <> Reply to Message

Introduce calls to BC code over the kernel to add
accounting of physical pages.

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

```
 fs/exec.c           |   15 +++-
 include/linux/rmap.h |    9 +-
 mm/fremap.c         |   11 ++
 mm/memory.c         |   52 ++++++++++++--
 mm/migrate.c        |   13 ++-
 mm/rmap.c           |   34 +++++++--
 mm/swapfile.c       |   15 +++-
 mm/vmscan.c         |  190 ++++++++++++++++++++++++++++++++++++++++++++++++++++
 8 files changed, 310 insertions(+), 29 deletions(-)

--- ./fs/exec.c.ve10 2006-11-07 11:56:20.000000000 +0300
+++ ./fs/exec.c 2006-11-07 11:57:23.000000000 +0300
@@ -51,6 +51,8 @@
```

```
 #include <linux/cn_proc.h>
 #include <linux/audit.h>

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

@@ -309,27 +311,34 @@ void install_arg_page(struct vm_area_str
  struct mm_struct *mm = vma->vm_mm;
  pte_t * pte;
  spinlock_t *ptl;
+ struct page_beancounter *pb;

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

+ if (bc_rsspage_prepare(page, vma, &pb))
+  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);
+ page_add_new_anon_rmap(page, vma, address, pb);
  pte_unmap_unlock(pte, ptl);

  /* no need for flush_tlb */
  return;
+
+out_unch:
+ bc_rsspage_release(pb);
 out:
  __free_page(page);
  force_sig(SIGKILL, current);
--- ./include/linux/rmap.h.ve10 2006-11-07 11:56:20.000000000 +0300
+++ ./include/linux/rmap.h 2006-11-07 11:57:23.000000000 +0300
```

```
@@ -69,9 +69,12 @@ void __anon_vma_link(struct vm_area_stru
 /*
  * rmap interfaces called when adding or removing pte of page
  */
-void page_add_anon_rmap(struct page *, struct vm_area_struct *, unsigned long);
-void page_add_new_anon_rmap(struct page *, struct vm_area_struct *, unsigned long);
-void page_add_file_rmap(struct page *);
+struct page_beancounter;
+void page_add_anon_rmap(struct page *, struct vm_area_struct *, unsigned long,
+  struct page_beancounter *);
+void page_add_new_anon_rmap(struct page *, struct vm_area_struct *,
+  unsigned long, struct page_beancounter *);
+void page_add_file_rmap(struct page *, struct page_beancounter *);
 void page_remove_rmap(struct page *);

 /**
--- ./mm/fremap.c.ve10 2006-11-07 11:56:20.000000000 +0300
+++ ./mm/fremap.c 2006-11-07 11:57:23.000000000 +0300
@@ -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 *pb;
+
+ if (bc_rsspage_prepare(page, vma, &pb))
+  goto out_nocharge;

  pte = get_locked_pte(mm, addr, &ptl);
  if (!pte)
@@ -81,13 +87,16 @@ int install_page(struct mm_struct *mm, s
  flush_icache_page(vma, page);
  pte_val = mk_pte(page, prot);
  set_pte_at(mm, addr, pte, pte_val);
- page_add_file_rmap(page);
+ page_add_file_rmap(page, pb);
  update_mmu_cache(vma, addr, pte_val);
  lazy_mmu_prot_update(pte_val);
  err = 0;
 unlock:
```

```
  pte_unmap_unlock(pte, ptl);
 out:
+ if (err != 0)
+  bc_rsspage_release(pb);
+out_nocharge:
  return err;
 }
 EXPORT_SYMBOL(install_page);
--- ./mm/memory.c.ve10 2006-11-07 11:56:20.000000000 +0300
+++ ./mm/memory.c 2006-11-07 11:57:23.000000000 +0300
@@ -57,6 +57,8 @@
 #include <asm/tlbflush.h>
 #include <asm/pgtable.h>

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

@@ -1119,7 +1121,7 @@ static int zeromap_pte_range(struct mm_s
   struct page *page = ZERO_PAGE(addr);
   pte_t zero_pte = pte_wrprotect(mk_pte(page, prot));
   page_cache_get(page);
-  page_add_file_rmap(page);
+  page_add_file_rmap(page, NULL);
   inc_mm_counter(mm, file_rss);
   BUG_ON(!pte_none(*pte));
   set_pte_at(mm, addr, pte, zero_pte);
@@ -1224,7 +1226,7 @@ static int insert_page(struct mm_struct
  /* Ok, finally just insert the thing.. */
  get_page(page);
  inc_mm_counter(mm, file_rss);
- page_add_file_rmap(page);
+ page_add_file_rmap(page, NULL);
  set_pte_at(mm, addr, pte, mk_pte(page, prot));

  retval = 0;
@@ -1485,6 +1487,7 @@ 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 *pb;

  old_page = vm_normal_page(vma, address, orig_pte);
  if (!old_page)
@@ -1570,6 +1573,9 @@ gotten:
   cow_user_page(new_page, old_page, address);
  }
```

```
+ if (bc_rsspage_prepare(new_page, vma, &pb))
+  goto oom;
+
  /*
   * Re-check the pte - we dropped the lock
   */
@@ -1597,12 +1603,14 @@ gotten:
   set_pte_at(mm, address, page_table, entry);
   update_mmu_cache(vma, address, entry);
   lru_cache_add_active(new_page);
-  page_add_new_anon_rmap(new_page, vma, address);
+  page_add_new_anon_rmap(new_page, vma, address, pb);

   /* Free the old page.. */
   new_page = old_page;
   ret |= VM_FAULT_WRITE;
- }
+ } else
+  bc_rsspage_release(pb);
+
  if (new_page)
   page_cache_release(new_page);
  if (old_page)
@@ -1979,6 +1987,7 @@ static int do_swap_page(struct mm_struct
  swp_entry_t entry;
  pte_t pte;
  int ret = VM_FAULT_MINOR;
+ struct page_beancounter *pb;

  if (!pte_unmap_same(mm, pmd, page_table, orig_pte))
   goto out;
@@ -2011,6 +2020,11 @@ static int do_swap_page(struct mm_struct
   count_vm_event(PGMAJFAULT);
  }

+ if (bc_rsspage_prepare(page, vma, &pb)) {
+  ret = VM_FAULT_OOM;
+  goto out;
+ }
+
  delayacct_clear_flag(DELAYACCT_PF_SWAPIN);
  mark_page_accessed(page);
  lock_page(page);
@@ -2024,6 +2038,7 @@ static int do_swap_page(struct mm_struct

  if (unlikely(!PageUptodate(page))) {
   ret = VM_FAULT_SIGBUS;
```

```
+  bc_rsspage_release(pb);
   goto out_nomap;
  }

@@ -2038,7 +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);
+ page_add_anon_rmap(page, vma, address, pb);

  swap_free(entry);
  if (vm_swap_full())
@@ -2060,6 +2075,7 @@ unlock:
 out:
  return ret;
 out_nomap:
+ bc_rsspage_release(pb);
  pte_unmap_unlock(page_table, ptl);
  unlock_page(page);
  page_cache_release(page);
@@ -2078,6 +2094,7 @@ static int do_anonymous_page(struct mm_s
  struct page *page;
  spinlock_t *ptl;
  pte_t entry;
+ struct page_beancounter *pb;

  if (write_access) {
   /* Allocate our own private page. */
@@ -2089,15 +2106,19 @@ static int do_anonymous_page(struct mm_s
   if (!page)
    goto oom;

+  if (bc_rsspage_prepare(page, vma, &pb))
+   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))
-   goto release;
+   goto release_pc;
+
   inc_mm_counter(mm, anon_rss);
   lru_cache_add_active(page);
-  page_add_new_anon_rmap(page, vma, address);
+  page_add_new_anon_rmap(page, vma, address, pb);
```

```
  } else {
   /* Map the ZERO_PAGE - vm_page_prot is readonly */
   page = ZERO_PAGE(address);
@@ -2109,7 +2130,7 @@ static int do_anonymous_page(struct mm_s
   if (!pte_none(*page_table))
    goto release;
   inc_mm_counter(mm, file_rss);
-  page_add_file_rmap(page);
+  page_add_file_rmap(page, NULL);
  }

  set_pte_at(mm, address, page_table, entry);
@@ -2120,9 +2141,14 @@ static int do_anonymous_page(struct mm_s
 unlock:
  pte_unmap_unlock(page_table, ptl);
  return VM_FAULT_MINOR;
+release_pc:
+ bc_rsspage_release(pb);
 release:
  page_cache_release(page);
  goto unlock;
+
+oom_release:
+ page_cache_release(page);
 oom:
  return VM_FAULT_OOM;
 }
@@ -2152,6 +2178,7 @@ static int do_no_page(struct mm_struct *
  int ret = VM_FAULT_MINOR;
  int anon = 0;
  struct page *dirty_page = NULL;
+ struct page_beancounter *pb;

  pte_unmap(page_table);
  BUG_ON(vma->vm_flags & VM_PFNMAP);
@@ -2209,6 +2236,9 @@ retry:
   }
  }

+ if (bc_rsspage_prepare(new_page, vma, &pb))
+  goto oom;
+
  page_table = pte_offset_map_lock(mm, pmd, address, &ptl);
  /*
   * For a file-backed vma, someone could have truncated or otherwise
@@ -2217,6 +2247,7 @@ retry:
   */
  if (mapping && unlikely(sequence != mapping->truncate_count)) {
```

```
   pte_unmap_unlock(page_table, ptl);
+  bc_rsspage_release(pb);
   page_cache_release(new_page);
   cond_resched();
   sequence = mapping->truncate_count;
@@ -2244,10 +2275,10 @@ retry:
   if (anon) {
    inc_mm_counter(mm, anon_rss);
    lru_cache_add_active(new_page);
-   page_add_new_anon_rmap(new_page, vma, address);
+   page_add_new_anon_rmap(new_page, vma, address, pb);
   } else {
    inc_mm_counter(mm, file_rss);
-   page_add_file_rmap(new_page);
+   page_add_file_rmap(new_page, pb);
    if (write_access) {
     dirty_page = new_page;
     get_page(dirty_page);
@@ -2255,6 +2286,7 @@ retry:
   }
  } else {
   /* One of our sibling threads was faster, back out. */
+  bc_rsspage_release(pb);
   page_cache_release(new_page);
   goto unlock;
  }
--- ./mm/migrate.c.ve10 2006-11-07 11:56:20.000000000 +0300
+++ ./mm/migrate.c 2006-11-07 11:57:23.000000000 +0300
@@ -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 *pb;

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

+ if (bc_rsspage_prepare(new, vma, &pb)) {
+  pte_unmap(ptep);
+  return;
+ }
+
  ptl = pte_lockptr(mm, pmd);
  spin_lock(ptl);
 pte = *ptep;
```

```
@@ -175,16 +181,19 @@ static void remove_migration_pte(struct
  set_pte_at(mm, addr, ptep, pte);

  if (PageAnon(new))
-  page_add_anon_rmap(new, vma, addr);
+  page_add_anon_rmap(new, vma, addr, pb);
  else
-  page_add_file_rmap(new);
+  page_add_file_rmap(new, pb);

  /* 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(pb);
 }

 /*
--- ./mm/rmap.c.ve10 2006-11-07 11:56:20.000000000 +0300
+++ ./mm/rmap.c 2006-11-07 11:57:51.000000000 +0300
@@ -48,6 +48,8 @@
 #include <linux/rcupdate.h>
 #include <linux/module.h>

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

 struct kmem_cache *anon_vma_cachep;
@@ -173,7 +175,8 @@ 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);
 }

 /*
@@ -522,14 +525,19 @@ static void __page_set_anon_rmap(struct
  * @page: the page to add the mapping to
  * @vma: the vm area in which the mapping is added
  * @address: the user virtual address mapped
+ * @pb:  the page beancounter to charge page with
```

```
  *
  * The caller needs to hold the pte lock.
  */
 void page_add_anon_rmap(struct page *page,
- struct vm_area_struct *vma, unsigned long address)
+ struct vm_area_struct *vma, unsigned long address,
+ struct page_beancounter *pb)
 {
- if (atomic_inc_and_test(&page->_mapcount))
+ if (atomic_inc_and_test(&page->_mapcount)) {
+  bc_rsspage_charge(pb);
    __page_set_anon_rmap(page, vma, address);
+ } else
+  bc_rsspage_release(pb);
   /* else checking page index and mapping is racy */
 }

@@ -538,27 +546,35 @@ void page_add_anon_rmap(struct page *pag
  * @page: the page to add the mapping to
  * @vma: the vm area in which the mapping is added
  * @address: the user virtual address mapped
+ * @pb:  the page beancounter to charge page with
  *
  * Same as page_add_anon_rmap but must only be called on *new* pages.
  * This means the inc-and-test can be bypassed.
  */
 void page_add_new_anon_rmap(struct page *page,
- struct vm_area_struct *vma, unsigned long address)
+ struct vm_area_struct *vma, unsigned long address,
+ struct page_beancounter *pb)
 {
  atomic_set(&page->_mapcount, 0); /* elevate count by 1 (starts at -1) */
+ bc_rsspage_charge(pb);
  __page_set_anon_rmap(page, vma, address);
 }

 /**
  * page_add_file_rmap - add pte mapping to a file page
- * @page: the page to add the mapping to
+ * @page: the page to add the mapping to
+ * @pb:  the page beancounter to charge page with
  *
  * The caller needs to hold the pte lock.
  */
-void page_add_file_rmap(struct page *page)
+void page_add_file_rmap(struct page *page, struct page_beancounter *pb)
 {
- if (atomic_inc_and_test(&page->_mapcount))
```

```
+ if (atomic_inc_and_test(&page->_mapcount)) {
+  if (pb)
+   bc_rsspage_charge(pb);
   __inc_zone_page_state(page, NR_FILE_MAPPED);
+ } else if (pb)
+  bc_rsspage_release(pb);
 }

 /**
@@ -569,6 +585,9 @@ void page_add_file_rmap(struct page *pag
  */
 void page_remove_rmap(struct page *page)
 {
+ struct page_beancounter *pb;
+
+ pb = page_pb(page);
  if (atomic_add_negative(-1, &page->_mapcount)) {
   if (unlikely(page_mapcount(page) < 0)) {
    printk (KERN_EMERG "Eeek! page_mapcount(page) went negative! (%d)\n",
page_mapcount(page));
@@ -578,6 +597,7 @@ void page_remove_rmap(struct page *page)
    BUG();
   }

+  bc_rsspage_uncharge(pb);
   /*
    * It would be tidy to reset the PageAnon mapping here,
    * but that might overwrite a racing page_add_anon_rmap
--- ./mm/swapfile.c.ve10 2006-11-07 11:56:20.000000000 +0300
+++ ./mm/swapfile.c 2006-11-07 11:57:23.000000000 +0300
@@ -28,6 +28,8 @@
 #include <linux/capability.h>
 #include <linux/syscalls.h>

+#include <bc/rsspages.h>
+
 #include <asm/pgtable.h>
 #include <asm/tlbflush.h>
 #include <linux/swapops.h>
@@ -501,13 +503,14 @@ unsigned int count_swap_pages(int type,
  * force COW, vm_page_prot omits write permission from any private vma.
  */
 static void unuse_pte(struct vm_area_struct *vma, pte_t *pte,
-  unsigned long addr, swp_entry_t entry, struct page *page)
+  unsigned long addr, swp_entry_t entry, struct page *page,
+  struct page_beancounter *pb)
 {
  inc_mm_counter(vma->vm_mm, anon_rss);
```

```
  get_page(page);
  set_pte_at(vma->vm_mm, addr, pte,
      pte_mkold(mk_pte(page, vma->vm_page_prot)));
- page_add_anon_rmap(page, vma, addr);
+ page_add_anon_rmap(page, vma, addr, pb);
  swap_free(entry);
  /*
   * Move the page to the active list so it is not
@@ -524,6 +527,10 @@ static int unuse_pte_range(struct vm_are
  pte_t *pte;
  spinlock_t *ptl;
  int found = 0;
+ struct page_beancounter *pb;
+
+ if (bc_rsspage_prepare(page, vma, &pb))
+  return 0;

  pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
  do {
@@ -532,12 +539,14 @@ static int unuse_pte_range(struct vm_are
    * Test inline before going to call unuse_pte.
    */
   if (unlikely(pte_same(*pte, swp_pte))) {
-   unuse_pte(vma, pte++, addr, entry, page);
+   unuse_pte(vma, pte++, addr, entry, page, pb);
    found = 1;
    break;
   }
  } while (pte++, addr += PAGE_SIZE, addr != end);
  pte_unmap_unlock(pte - 1, ptl);
+ if (!found)
+  bc_rsspage_release(pb);
  return found;
 }

--- ./mm/vmscan.c.ve10 2006-11-07 11:56:20.000000000 +0300
+++ ./mm/vmscan.c 2006-11-07 11:57:23.000000000 +0300
@@ -39,6 +39,8 @@
 #include <linux/kthread.h>
 #include <linux/freezer.h>

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

@@ -1103,6 +1105,194 @@ 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)
+   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 12/13] BC: numtasks accounting
Posted by dev on Thu, 09 Nov 2006 17:02:17 GMT
View Forum Message <> Reply to Message

Beautifull tasks accounting/limiting beancounter control.

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

```
 include/linux/sched.h |    1 +
 kernel/bc/misc.c      |   31 ++++++++++++++++++++++++++++++++
 2 files changed, 32 insertions(+)

--- ./include/linux/sched.h.bctasks 2006-11-03 17:47:38.000000000 +0300
+++ ./include/linux/sched.h 2006-11-03 17:51:57.000000000 +0300
@@ -1068,6 +1068,7 @@ int copy_beancounter(struct task_struct
 #endif
 #ifdef CONFIG_BEANCOUNTERS
  struct beancounter *exec_bc;
+ struct beancounter *task_bc;
 #endif
 };

--- ./kernel/bc/misc.c.bctasks 2006-11-03 17:47:38.000000000 +0300
+++ ./kernel/bc/misc.c 2006-11-03 17:51:57.000000000 +0300
@@ -17,6 +17,10 @@ int copy_beancounter(struct task_struct
  struct beancounter *bc;

  bc = parent->exec_bc;
+ if (bc_charge(bc, BC_NUMTASKS, 1, BC_LIMIT))
+  return -ENOMEM;
+
+ tsk->task_bc = bc_get(bc);
  tsk->exec_bc = bc_get(bc);
  return 0;
 }
@@ -25,6 +29,10 @@ void free_beancounter(struct task_struct
 {
  struct beancounter *bc;

+ bc = tsk->task_bc;
+ bc_uncharge(bc, BC_NUMTASKS, 1);
+ bc_put(bc);
```

```
+
  bc = tsk->exec_bc;
  bc_put(bc);
 }
@@ -86,3 +94,26 @@ int bc_task_move(struct task_struct *tsk
  return err;
 }
 EXPORT_SYMBOL(bc_task_move);
+
+#define BC_NUMTASKS_BARRIER 128
+#define BC_NUMTASKS_LIMIT 128
+
+static int bc_task_init(struct beancounter *bc, int i)
+{
+ bc_init_resource(&bc->bc_parms[BC_NUMTASKS],
+    BC_NUMTASKS_BARRIER, BC_NUMTASKS_LIMIT);
+ return 0;
+}
+
+static struct bc_resource bc_task_resource = {
+ .bcr_name = "numtasks",
+ .bcr_init = bc_task_init,
+};
+
+static int __init bc_misc_init_resource(void)
+{
+ bc_register_resource(BC_NUMTASKS, &bc_task_resource);
+ return 0;
+}
+
+__initcall(bc_misc_init_resource);
```

Subject: [PATCH 13/13] BC: numfiles accounting
Posted by dev on Thu, 09 Nov 2006 17:03:02 GMT
View Forum Message <> Reply to Message

Very simple beancounter for accounting and limiting
container number of opened files.

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

```
 fs/file_table.c    |   12 ++++++++++-
 include/bc/misc.h  |   27 +++++++++++++++++++++++++++
 include/linux/fs.h |    3 +++
```

```
 kernel/bc/misc.c   |   41 ++++++++++++++++++++++++++++++++++++++++++++++
 4 files changed, 82 insertions(+), 1 deletion(-)

--- ./fs/file_table.c.bcnumfiles 2006-11-09 11:29:11.000000000 +0300
+++ ./fs/file_table.c 2006-11-09 11:35:34.000000000 +0300
@@ -23,6 +23,8 @@
 #include <linux/sysctl.h>
 #include <linux/percpu_counter.h>

+#include <bc/misc.h>
+
 #include <asm/atomic.h>

 /* sysctl tunables... */
@@ -44,6 +46,7 @@ static inline void file_free_rcu(struct
 static inline void file_free(struct file *f)
 {
  percpu_counter_dec(&nr_files);
+ bc_file_uncharge(f);
  call_rcu(&f->f_u.fu_rcuhead, file_free_rcu);
 }

@@ -108,8 +111,11 @@ struct file *get_empty_filp(void)
  if (f == NULL)
   goto fail;

- percpu_counter_inc(&nr_files);
  memset(f, 0, sizeof(*f));
+ if (bc_file_charge(f))
+  goto fail_charge;
+
+ percpu_counter_inc(&nr_files);
  if (security_file_alloc(f))
   goto fail_sec;

@@ -136,6 +142,10 @@ fail_sec:
  file_free(f);
 fail:
  return NULL;
+
+fail_charge:
+ kmem_cache_free(filp_cachep, f);
+ return NULL;
 }

 EXPORT_SYMBOL(get_empty_filp);
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/misc.h 2006-11-09 11:34:42.000000000 +0300
```

```
@@ -0,0 +1,27 @@
+/*
+ * include/bc/misc.h
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#ifndef __BC_MISC_H__
+#define __BC_MISC_H__
+
+struct file;
+
+#ifdef CONFIG_BEANCOUNTERS
+int __must_check bc_file_charge(struct file *);
+void bc_file_uncharge(struct file *);
+#else
+static inline int __must_check bc_file_charge(struct file *f)
+{
+ return 0;
+}
+
+static inline void bc_file_uncharge(struct file *f)
+{
+}
+#endif
+
+#endif
--- ./include/linux/fs.h.bcnumfiles 2006-11-09 11:29:12.000000000 +0300
+++ ./include/linux/fs.h 2006-11-09 11:34:42.000000000 +0300
@@ -802,6 +802,9 @@ struct file {
  struct kevent_storage st;
 #endif
  struct address_space *f_mapping;
+#ifdef CONFIG_BEANCOUNTERS
+ struct beancounter *f_bc;
+#endif
 };
 extern spinlock_t files_lock;
 #define file_list_lock() spin_lock(&files_lock);
--- ./kernel/bc/misc.c.bcnumfiles 2006-11-09 11:34:31.000000000 +0300
+++ ./kernel/bc/misc.c 2006-11-09 11:34:42.000000000 +0300
@@ -7,6 +7,7 @@
 #include <linux/sched.h>
 #include <linux/stop_machine.h>
 #include <linux/module.h>
+#include <linux/fs.h>
```

```
 #include <bc/beancounter.h>
 #include <bc/task.h>
@@ -95,6 +96,45 @@ int bc_task_move(struct task_struct *tsk
 }
 EXPORT_SYMBOL(bc_task_move);

+int bc_file_charge(struct file *file)
+{
+ int sev;
+ struct beancounter *bc;
+
+ bc = get_exec_bc();
+ sev = (capable(CAP_SYS_ADMIN) ? BC_LIMIT : BC_BARRIER);
+
+ if (bc_charge(bc, BC_NUMFILES, 1, sev))
+  return -EMFILE;
+
+ file->f_bc = bc_get(bc);
+ return 0;
+}
+
+void bc_file_uncharge(struct file *file)
+{
+ struct beancounter *bc;
+
+ bc = file->f_bc;
+ bc_uncharge(bc, BC_NUMFILES, 1);
+ bc_put(bc);
+}
+
+#define BC_NUMFILES_BARRIER 256
+#define BC_NUMFILES_LIMIT 512
+
+static int bc_files_init(struct beancounter *bc, int i)
+{
+ bc_init_resource(&bc->bc_parms[BC_NUMFILES],
+   BC_NUMFILES_BARRIER, BC_NUMFILES_LIMIT);
+ return 0;
+}
+
+static struct bc_resource bc_files_resource = {
+ .bcr_name = "numfiles",
+ .bcr_init = bc_files_init,
+};
+
 #define BC_NUMTASKS_BARRIER 128
 #define BC_NUMTASKS_LIMIT 128
```

```
@@ -113,6 +153,7 @@ static struct bc_resource bc_task_resour
 static int __init bc_misc_init_resource(void)
 {
  bc_register_resource(BC_NUMTASKS, &bc_task_resource);
+ bc_register_resource(BC_NUMFILES, &bc_files_resource);
  return 0;
 }
```

Subject: Re: [PATCH 6/13] BC: kmemsize accounting (core)
Posted by Paul Jackson on Thu, 09 Nov 2006 19:05:26 GMT
View Forum Message <> Reply to Message

> +#define __GFP_BC ((__force gfp_t)0x80000u) /* Charge allocation with BC */

Please include the term "beancounter" in that comment.

--

I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.925.600.0401

Subject: Re: [PATCH 1/13] BC: atomic_dec_and_lock_irqsave() helper
Posted by Cedric Le Goater on Fri, 10 Nov 2006 15:19:37 GMT
View Forum Message <> Reply to Message

Hello Kirill, Hello Pavel,

Kirill Korotaev wrote:
> 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().

You should probably send that one independently from the BC
patchset.

C.

Subject: Re: [PATCH 1/13] BC: atomic_dec_and_lock_irqsave() helper
Posted by dev on Fri, 10 Nov 2006 16:40:31 GMT
View Forum Message <> Reply to Message

> Hello Kirill, Hello Pavel,
>
> Kirill Korotaev wrote:
>
>>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().
>
>
> You should probably send that one independently from the BC
> patchset.
Maybe, but BCs are the only user of this so far...

Thanks,
Kirill

Subject: Re: [PATCH 6/13] BC: kmemsize accounting (core)
Posted by Pekka Enberg on Fri, 10 Nov 2006 22:46:36 GMT
View Forum Message <> Reply to Message

Hi,

On 11/9/06, Kirill Korotaev <dev@sw.ru> wrote:
> +#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;

Why do we want to track each allocated _object_ in the slab? Isn't
tracking pages enough?

Subject: Re: [PATCH 6/13] BC: kmemsize accounting (core)
Posted by Pekka Enberg on Fri, 10 Nov 2006 22:50:40 GMT
View Forum Message <> Reply to Message

On 11/9/06, Kirill Korotaev <dev@sw.ru> wrote:
> +#ifdef CONFIG_BEANCOUNTERS
> +#define BC_EXTRASIZE	sizeof(struct beancounter *)

Would much prefer you put all beancounter stuff into one #ifdef block
to avoid clutter.

> @@ -2579,14 +2635,14 @@ 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

No #ifdef within functions, please, but instead, make it an static
inline function.

Subject: Re: [PATCH 6/13] BC: kmemsize accounting (core)
Posted by Pavel Machek on Sat, 11 Nov 2006 05:50:50 GMT
View Forum Message <> Reply to Message

Hi!

> --- /dev/null 2006-07-18 14:52:43.075228448 +0400
> +++ ./include/bc/kmem.h 2006-11-03 15:48:26.000000000 +0300
> @@ -0,0 +1,48 @@
> +/*
> + * include/bc/kmem.h
> + *
> + * Copyright (C) 2006 OpenVZ SWsoft Inc

> + *
> + */

GPL would be nice, as would be email address of someone who worked on
this file.

> --- /dev/null 2006-07-18 14:52:43.075228448 +0400
> +++ ./kernel/bc/kmem.c 2006-11-03 15:48:26.000000000 +0300
> @@ -0,0 +1,112 @@
> +/*
> + * kernel/bc/kmem.c
> + *
> + * Copyright (C) 2006 OpenVZ SWsoft Inc
> + *
> + */

Same here.

> +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;

You can do this before if() and spare a dereference.

	Pavel
--
Thanks for all the (sleeping) penguins.

Subject: Re: [PATCH 6/13] BC: kmemsize accounting (core)
Posted by Pavel Emelianov on Mon, 13 Nov 2006 12:13:41 GMT
View Forum Message <> Reply to Message

Pekka Enberg wrote:
> Hi,
>
> On 11/9/06, Kirill Korotaev <dev@sw.ru> wrote:
>> +#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;
>
> Why do we want to track each allocated _object_ in the slab? Isn't
> tracking pages enough?

No. One page may contain objects allocated in different beancounters.

Subject: Re: BC: resource beancounters (v6) (with userpages reclamation + configfs)
Posted by Herbert Poetzl on Sun, 19 Nov 2006 19:41:28 GMT
View Forum Message <> Reply to Message

On Thu, Nov 09, 2006 at 07:49:28PM +0300, Kirill Korotaev wrote:
> MAJOR CHANGES in v6 (see details below):
> - configfs interface instead of syscalls (as wanted by CKRM people...)
> - added numfiles resource accounting
> - added numtasks resource accounting
>
> numfiles and numtasks controllers demonstrate how
> clean and simple BC interface is.
>
> Patch set is applicable to 2.6.19-rc5-mm1
>
> ----------
>
> Resource BeanCounters (BC).
>
> BC allows to account and control consumption
> of kernel resources used by *group* of processes
> (users, containers, ...).
>
> Draft BC 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 and not reclaimable should be accounted and
> limited for DoS protection.
> e.g. page tables, task structs, vmas etc.

kernel memory is not accounted on a linux system
right now, and user can probably quite easily use
that for DoS ... shouldn't that become a separate
user limit in the first place?

> - 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).

how much overhead does this add to the memory/page
management? do we really want to account shared pages
at all, if so, why is accounting a fraction to each
group the 'proper' way to do so? (would expect to
account it to all of them equally)

IMHO it would be sufficient to extend existing page
accounting to a 'memory' namespace isntead of adding
another complex mechanism like the beancounters

> - 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)
>
> Summary of changes from v5 patch set:
> * configfs interface instead of syscalls (as wanted by CKRM people)
> * added numfiles resource accounting
> * added numtasks resource accounting
> * introduced dummy_resource to handle case when
>   no resource registered
> * calls to rss accounting are integrated to rmap calls

again, how much overhead does this add?
can you provide some numbers/tests here?

TIA,
Herbert

> 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.
>
> Thanks,
> Kirill
> -
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/

Subject: Re: [ckrm-tech] [PATCH 4/13] BC: context handling
Posted by Paul Menage on Thu, 23 Nov 2006 07:48:56 GMT
View Forum Message <> Reply to Message

On 11/9/06, Kirill Korotaev <dev@sw.ru> wrote:
> +
> +int bc_task_move(int pid, struct beancounter *bc, int whole)
> +{

...

> +
> +	down_write(&mm->mmap_sem);
> +	err = stop_machine_run(do_set_bcid, &data, NR_CPUS);
> +	up_write(&mm->mmap_sem);

Isn't this a little heavyweight for moving a task into/between beancounters?

Paul

Subject: Re: [ckrm-tech] [PATCH 4/13] BC: context handling
Posted by Pavel Emelianov on Thu, 23 Nov 2006 08:35:10 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On 11/9/06, Kirill Korotaev <dev@sw.ru> wrote:
>> +
>> +int bc_task_move(int pid, struct beancounter *bc, int whole)
>> +{
>
> ...
>
>> +
>> +	down_write(&mm->mmap_sem);

>> +	err = stop_machine_run(do_set_bcid, &data, NR_CPUS);
>> +	up_write(&mm->mmap_sem);
>
> Isn't this a little heavyweight for moving a task into/between
> beancounters?

It's a main reason we were against moving arbitrary task.

We need to track the situation when we change beancounter on
task that is currently handles an interrupt and thus set a
temporary BC as exec one. I see no other way that keeps pair
set_exec_bc()/get_exec_bc() lock-less.

The problem is even larger than I've described. set_exec_bc()
is used widely in OpenVZ beancounters to set temporary context
e.g. for skb handling. Thus we need some safe way to "catch"
the task in a "safe" place. In OpenVZ we solve this by moving
only current into beancounter. In this patch set we have to
move arbitrary task and thus - such complication.

I repeat - we can do this w/o stop_machine, but this would
require locking in set_exec_bc()/get_exec_bc() but it's too
bad. Moving tasks happens rarely but setting context is a
very common operation (e.g. in each interrupt).

We can do the following:

```
 if (tsk == current)
    /* fast way */
    tsk->exec_bc = bc;
 else
    /* slow way */
    stop_machine_run(...);
```

What do you think?

> Paul
> -
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/
>

Subject: Re: [ckrm-tech] [PATCH 4/13] BC: context handling
Posted by Paul Menage on Thu, 23 Nov 2006 08:53:51 GMT

On 11/23/06, Pavel Emelianov <xemul@openvz.org> wrote:
>
> We can do the following:
>
> if (tsk == current)
> /* fast way */
> tsk->exec_bc = bc;
> else
> /* slow way */
> stop_machine_run(...);
>
> What do you think?

How about having two pointers per task:

- exec_bc, which is the one used for charging
- real_bc, which is the task's actual beancounter

at the start of irq, do

current->exec_bc = &init_bc;

at the end of irq, do

current->exec_bc = current->real_bc;

When moving a task to a different bc do:

task->real_bc = new_bc;
atomic_cmpxchg(&task->exec_bc, old_bc, new_bc);

(with appropriate memory barriers). So if the task is in an irq with a
modified exec_bc pointer, we do nothing, otherwise we update exec_bc
to point to the new real_bc.

Paul

Subject: Re: [ckrm-tech] [PATCH 4/13] BC: context handling
Posted by Pavel Emelianov on Thu, 23 Nov 2006 09:20:29 GMT

Paul Menage wrote:
> On 11/23/06, Pavel Emelianov <xemul@openvz.org> wrote:
>>
>> We can do the following:

>>
>>   if (tsk == current)
>>   /* fast way */
>>   tsk->exec_bc = bc;
>>   else
>>   /* slow way */
>>   stop_machine_run(...);
>>
>> What do you think?
>
> How about having two pointers per task:
>
> - exec_bc, which is the one used for charging
> - real_bc, which is the task's actual beancounter
>
> at the start of irq, do
>
> current->exec_bc = &init_bc;
>
> at the end of irq, do
>
> current->exec_bc = current->real_bc;
>
> When moving a task to a different bc do:
>
> task->real_bc = new_bc;
> atomic_cmpxchg(&task->exec_bc, old_bc, new_bc);

You mean moving is like this:

old_bc = task->real_bc;
task->real_bc = new_bc;
cmpxchg(&tsk->exec_bc, old_bc, new_bc);

? Then this won't work:

Initialisation:
current->exec_bc = init_bc;
current->real_bc = init_bc;
...
IRQ:
current->exec_bc = init_bc;
...
                    old_bc = tsk->real_bc; /* init_bc */
                    tsk->real_bc = bc1;
                    cx(tsk->exec_bc, init_bc, bc1); /* ok */
...
Here at the middle of an interrupt

we have bc1 set as exec_bc on task
which IS wrong!
...
current->exec_bc =
	current->real_bc;

We need some way to be sure that task isn't running at
the moment we change it's beancounter. Otherwise we're
risking that we'll spoil some temporary context.

> (with appropriate memory barriers). So if the task is in an irq with a
> modified exec_bc pointer, we do nothing, otherwise we update exec_bc
> to point to the new real_bc.
>
> Paul

Subject: Re: [ckrm-tech] [PATCH 4/13] BC: context handling
Posted by Paul Menage on Thu, 23 Nov 2006 09:31:47 GMT
View Forum Message <> Reply to Message

On 11/23/06, Pavel Emelianov <xemul@openvz.org> wrote:
> You mean moving is like this:
>
> old_bc = task->real_bc;
> task->real_bc = new_bc;
> cmpxchg(&tsk->exec_bc, old_bc, new_bc);
>
> ? Then this won't work:
>
> Initialisation:
> current->exec_bc = init_bc;
> current->real_bc = init_bc;
> ...
> IRQ:
> current->exec_bc = init_bc;
> ...
>		old_bc = tsk->real_bc; /* init_bc */
>		tsk->real_bc = bc1;
>		cx(tsk->exec_bc, init_bc, bc1); /* ok */
> ...
> Here at the middle of an interrupt
> we have bc1 set as exec_bc on task
> which IS wrong!

You could get round that by having a separate "irq_bc" that's never
valid for a task not in an interrupt.

Paul

Subject: Re: [ckrm-tech] [PATCH 4/13] BC: context handling
Posted by Pavel Emelianov on Thu, 23 Nov 2006 09:56:00 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On 11/23/06, Pavel Emelianov <xemul@openvz.org> wrote:
>> You mean moving is like this:
>>
>> old_bc = task->real_bc;
>> task->real_bc = new_bc;
>> cmpxchg(&tsk->exec_bc, old_bc, new_bc);
>>
>> ? Then this won't work:
>>
>> Initialisation:
>> current->exec_bc = init_bc;
>> current->real_bc = init_bc;
>> ...
>> IRQ:
>> current->exec_bc = init_bc;
>> ...
>> old_bc = tsk->real_bc; /* init_bc */
>> tsk->real_bc = bc1;
>> cx(tsk->exec_bc, init_bc, bc1); /* ok */
>> ...
>> Here at the middle of an interrupt
>> we have bc1 set as exec_bc on task
>> which IS wrong!
>
> You could get round that by having a separate "irq_bc" that's never
> valid for a task not in an interrupt.

No no no. This is not what is needed. You see, we do have to
set exec_bc as temporary (and atomic) context. Having temporary
context is 1. flexible 2. needed by beancounters' network accountig.
We have to track this particular scenario.

Moreover making get_exec_bc() as
if (in_interrupt())
 return &irq_bc;
else
 return current->exec_bc;
is awful. It must me simple and stupid to allow us making temporary
contexts in any place of code.

Maybe we can make smth similar to wait_task_inactive and change
it's beancounter before unlocking the runqueue?

> Paul
>

Subject: Re: [ckrm-tech] [PATCH 4/13] BC: context handling
Posted by Paul Menage on Thu, 23 Nov 2006 10:18:46 GMT
View Forum Message <> Reply to Message

On 11/23/06, Pavel Emelianov <xemul@openvz.org> wrote:
> Paul Menage wrote:
> > On 11/23/06, Pavel Emelianov <xemul@openvz.org> wrote:
> >> You mean moving is like this:
> >>
> >> old_bc = task->real_bc;
> >> task->real_bc = new_bc;
> >> cmpxchg(&tsk->exec_bc, old_bc, new_bc);
> >>
> >> ? Then this won't work:
> >>
> >> Initialisation:
> >> current->exec_bc = init_bc;
> >> current->real_bc = init_bc;
> >> ...
> >> IRQ:
> >> current->exec_bc = init_bc;
> >> ...
> >>   old_bc = tsk->real_bc; /* init_bc */
> >>   tsk->real_bc = bc1;
> >>   cx(tsk->exec_bc, init_bc, bc1); /* ok */
> >> ...
> >> Here at the middle of an interrupt
> >> we have bc1 set as exec_bc on task
> >> which IS wrong!
> >
> > You could get round that by having a separate "irq_bc" that's never
> > valid for a task not in an interrupt.
>
> No no no. This is not what is needed. You see, we do have to
> set exec_bc as temporary (and atomic) context. Having temporary
> context is 1. flexible 2. needed by beancounters' network accountig.

I don't see why having an irq_bc wouldn't solve this. At the start of
the interrupt handler, set current->exec_bc to &irq_bc; at the end set
it to current->real_bc; use the cmpxchg() that I suggested to ensure
that you never update task->exec_bc from another task if it's not

equal to task->real_bc; use RCU to ensure that a beancounter is never
freed while someone might be accessing it.

>
> Maybe we can make smth similar to wait_task_inactive and change
> it's beancounter before unlocking the runqueue?

That could work too.

Paul

Subject: Re: [ckrm-tech] [PATCH 4/13] BC: context handling
Posted by Pavel Emelianov on Thu, 23 Nov 2006 10:45:23 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On 11/23/06, Pavel Emelianov <xemul@openvz.org> wrote:
>> Paul Menage wrote:
>> > On 11/23/06, Pavel Emelianov <xemul@openvz.org> wrote:
>> >> You mean moving is like this:
>> >>
>> >> old_bc = task->real_bc;
>> >> task->real_bc = new_bc;
>> >> cmpxchg(&tsk->exec_bc, old_bc, new_bc);
>> >>
>> >> ? Then this won't work:
>> >>
>> >> Initialisation:
>> >> current->exec_bc = init_bc;
>> >> current->real_bc = init_bc;
>> >> ...
>> >> IRQ:
>> >> current->exec_bc = init_bc;
>> >> ...
>> >>   old_bc = tsk->real_bc; /* init_bc */
>> >>   tsk->real_bc = bc1;
>> >>   cx(tsk->exec_bc, init_bc, bc1); /* ok */
>> >> ...
>> >> Here at the middle of an interrupt
>> >> we have bc1 set as exec_bc on task
>> >> which IS wrong!
>> >
>> > You could get round that by having a separate "irq_bc" that's never
>> > valid for a task not in an interrupt.
>>
>> No no no. This is not what is needed. You see, we do have to
>> set exec_bc as temporary (and atomic) context. Having temporary

>> context is 1. flexible 2. needed by beancounters' network accountig.
>
> I don't see why having an irq_bc wouldn't solve this. At the start of
> the interrupt handler, set current->exec_bc to &irq_bc; at the end set
> it to current->real_bc; use the cmpxchg() that I suggested to ensure
> that you never update task->exec_bc from another task if it's not
> equal to task->real_bc; use RCU to ensure that a beancounter is never
> freed while someone might be accessing it.

Oh, I see. I just didn't get your idea. This will work, but
1. we separate interrupt accounting from all the others'
2. for interrupts only. In case we want to set init_bc as
   temporary context all will be broken...

We need some generic solution independent from what
exactly is set as temporary exec_bc.

>>
>> Maybe we can make smth similar to wait_task_inactive and change
>> it's beancounter before unlocking the runqueue?
>
> That could work too.

Could work, but whether everyone will like such intrusion...
I agree that stop_machine isn't nicer. This is a temporary
solution that works for sure. Better one will follow...

Subject: Re: [ckrm-tech] [PATCH 4/13] BC: context handling
Posted by Pavel Emelianov on Fri, 24 Nov 2006 10:10:33 GMT
View Forum Message <> Reply to Message

I've got it! That's what will work:

```
struct task_struct {
 ...
 struct beancounter *exec_bc;
 struct beancounter *tmp_exec_bc; /* is set to NULL on
                                   * tsk creation
                                   */
};

struct beancounter get_exec_bc(void)
{
 if (current->tmp_exec_bc)
  return current->tmp_exec_bc;
 return rcu_dereference(current->exec_bc);
}
```

```
struct beancounter set_tmp_exec_bc(struct beancounter *new)
{
 struct beancounter *old;

 old = current->tmp_exec_bc;
 current->tmp_exec_bc = new;
 return old;
}

void reset_tmp_exec_bc(struct beancounter *expected_old)
{
 BUG_ON(current->tmp_exec_bc != expected_old);
 current->tmp_exec_bc = NULL;
}

void move_task(struct task_struct *tsk, struct beancounter *bc)
{
 struct beancounter *old;

 mutex_lock(&tsk_move_mutex);
 old = tsk->exec_bc;
 get_bc(bc);
 rcu_assign_pointer(current->exec_bc, bc);
 syncronize_rcu();
 mutex_unlock(&tsk_move_mutex);

 bc_put(old);
}
```

I will implement this in the next beancounter patches.
Thanks for discussion :)

Subject: Re: [ckrm-tech] [PATCH 4/13] BC: context handling
Posted by Paul Menage on Sat, 25 Nov 2006 00:09:03 GMT
View Forum Message <> Reply to Message

On 11/24/06, Pavel Emelianov <xemul@openvz.org> wrote:
> I've got it! That's what will work:
>
> struct task_struct {
>   ...
>   struct beancounter *exec_bc;
>   struct beancounter *tmp_exec_bc; /* is set to NULL on
>   * tsk creation
>   */

> };
>
> struct beancounter get_exec_bc(void)
> {
> if (current->tmp_exec_bc)
> return current->tmp_exec_bc;
> return rcu_dereference(current->exec_bc);
> }

Don't forget that this means all callers need to be in an
rcu_read_lock() section.

>
> I will implement this in the next beancounter patches.

This is looking remarkably like the mechanism in use for my generic
containers patches (inherited from Paul Jackson's cpusets code). In
the last set of patches that I posted on Wednesday night, I included
the example of the beancounters core and numfiles counter implemented
on top of the generic containers - basically pulling out the hash
table, refcounting and most of the configfs code (since that's handled
by the generic containers), and moving the attribute management
configfs code to the use the containerfs filesystem interface instead.
The rest is pretty much unchanged.

I think you could continue to use the tmp_exec_bc idea with this, and
have get_exec_bc() use the tmp_exec_bc if it existed, or else get the
bc pointer via the container system.

I'd appreciate any feedback you had on that approach.

Paul

Subject: Re: [ckrm-tech] [PATCH 4/13] BC: context handling
Posted by Pavel Emelianov on Mon, 27 Nov 2006 08:27:41 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On 11/24/06, Pavel Emelianov <xemul@openvz.org> wrote:
>> I've got it! That's what will work:
>>
>> struct task_struct {
>> ...
>> struct beancounter *exec_bc;
>> struct beancounter *tmp_exec_bc; /* is set to NULL on
>> * tsk creation
>> */

>> };
>>
>> struct beancounter get_exec_bc(void)
>> {
>>   if (current->tmp_exec_bc)
>>   return current->tmp_exec_bc;
>>   return rcu_dereference(current->exec_bc);
>> }
>
> Don't forget that this means all callers need to be in an
> rcu_read_lock() section.

Sure. This is done for these particular cases.

>>
>> I will implement this in the next beancounter patches.
>
> This is looking remarkably like the mechanism in use for my generic
> containers patches (inherited from Paul Jackson's cpusets code). In
> the last set of patches that I posted on Wednesday night, I included
> the example of the beancounters core and numfiles counter implemented
> on top of the generic containers - basically pulling out the hash
> table, refcounting and most of the configfs code (since that's handled
> by the generic containers), and moving the attribute management
> configfs code to the use the containerfs filesystem interface instead.
> The rest is pretty much unchanged.
>
> I think you could continue to use the tmp_exec_bc idea with this, and
> have get_exec_bc() use the tmp_exec_bc if it existed, or else get the
> bc pointer via the container system.

I'll look through your patches this week and send my opinion.

> I'd appreciate any feedback you had on that approach.
>
> Paul