Subject: [PATCH] BC: resource beancounters (v2)
Posted by dev on Wed, 23 Aug 2006 10:44:18 GMT
View Forum Message <> Reply to Message

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

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

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

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

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

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

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

The patches in these series are:
diff-bc-kconfig.patch:
  Adds kernel/bc/Kconfig file with UBC options and
  includes it into arch Kconfigs

diff-bc-core.patch:
  Contains core functionality and interfaces of BC:

find/create beancounter, initialization,
charge/uncharge of resource, core objects' declarations.

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

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

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

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

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

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

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

Summary of changes from 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

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

Patch set is applicable to 2.6.18-rc4-mm2

Thanks,
Kirill

Subject: [PATCH 1/6] BC: kconfig
Posted by dev on Wed, 23 Aug 2006 10:59:31 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>

---

 arch/i386/Kconfig | 2 ++
 arch/ia64/Kconfig | 2 ++
 arch/powerpc/Kconfig | 2 ++
 arch/ppc/Kconfig | 2 ++
 arch/sparc/Kconfig | 2 ++
 arch/sparc64/Kconfig | 2 ++
 arch/x86_64/Kconfig | 2 ++
 kernel/bc/Kconfig | 25 +++++++++++++++++++++++++
 8 files changed, 39 insertions(+)

--- ./arch/i386/Kconfig.bckm 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/i386/Kconfig 2006-07-28 14:10:41.000000000 +0400
@@ -1146,6 +1146,8 @@ source "crypto/Kconfig"

 source "lib/Kconfig"

+source "kernel/bc/Kconfig"

```
+
 #
 # Use the generic interrupt handling code in kernel/irq/:
 #
--- ./arch/ia64/Kconfig.bckm 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/ia64/Kconfig 2006-07-28 14:10:56.000000000 +0400
@@ -481,6 +481,8 @@ source "fs/Kconfig"

 source "lib/Kconfig"

+source "kernel/bc/Kconfig"
+
 #
 # Use the generic interrupt handling code in kernel/irq/:
 #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>

---

 arch/i386/Kconfig    |    2 ++
 arch/ia64/Kconfig    |    2 ++
 arch/powerpc/Kconfig |    2 ++
 arch/ppc/Kconfig     |    2 ++
 arch/sparc/Kconfig   |    2 ++
 arch/sparc64/Kconfig |    2 ++
 arch/x86_64/Kconfig  |    2 ++
 kernel/bc/Kconfig    |   25 +++++++++++++++++++++++++
 8 files changed, 39 insertions(+)

--- ./arch/i386/Kconfig.bckm 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/i386/Kconfig 2006-07-28 14:10:41.000000000 +0400
@@ -1146,6 +1146,8 @@ source "crypto/Kconfig"

 source "lib/Kconfig"

+source "kernel/bc/Kconfig"
+
 #
 # Use the generic interrupt handling code in kernel/irq/:
 #
--- ./arch/ia64/Kconfig.bckm 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/ia64/Kconfig 2006-07-28 14:10:56.000000000 +0400
@@ -481,6 +481,8 @@ source "fs/Kconfig"

 source "lib/Kconfig"
```

```
+source "kernel/bc/Kconfig"
+
 #
 # Use the generic interrupt handling code in kernel/irq/:
 #
--- ./arch/powerpc/Kconfig.arkcfg 2006-08-07 14:07:12.000000000 +0400
+++ ./arch/powerpc/Kconfig 2006-08-10 17:55:58.000000000 +0400
@@ -1038,6 +1038,8 @@ source "arch/powerpc/platforms/iseries/K

 source "lib/Kconfig"

+source "kernel/bc/Kconfig"
+
 menu "Instrumentation Support"
        depends on EXPERIMENTAL

--- ./arch/ppc/Kconfig.arkcfg 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/ppc/Kconfig 2006-08-10 17:56:13.000000000 +0400
@@ -1414,6 +1414,8 @@ endmenu

 source "lib/Kconfig"

+source "kernel/bc/Kconfig"
+
 source "arch/powerpc/oprofile/Kconfig"

 source "arch/ppc/Kconfig.debug"
--- ./arch/sparc/Kconfig.arkcfg 2006-04-21 11:59:32.000000000 +0400
+++ ./arch/sparc/Kconfig 2006-08-10 17:56:24.000000000 +0400
@@ -296,3 +296,5 @@ source "security/Kconfig"
 source "crypto/Kconfig"

 source "lib/Kconfig"
+
+source "kernel/bc/Kconfig"
--- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
+++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
@@ -432,3 +432,5 @@ source "security/Kconfig"
 source "crypto/Kconfig"

 source "lib/Kconfig"
+
+source "kernel/bc/Kconfig"
--- ./arch/x86_64/Kconfig.bckm 2006-07-10 12:39:11.000000000 +0400
+++ ./arch/x86_64/Kconfig 2006-07-28 14:10:49.000000000 +0400
@@ -655,3 +655,5 @@ source "security/Kconfig"
 source "crypto/Kconfig"
```

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

--- ./arch/powerpc/Kconfig.arkcfg 2006-08-07 14:07:12.000000000 +0400
+++ ./arch/powerpc/Kconfig 2006-08-10 17:55:58.000000000 +0400
@@ -1038,6 +1038,8 @@ source "arch/powerpc/platforms/iseries/K

 source "lib/Kconfig"

+source "kernel/bc/Kconfig"
+
 menu "Instrumentation Support"
         depends on EXPERIMENTAL

--- ./arch/ppc/Kconfig.arkcfg 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/ppc/Kconfig 2006-08-10 17:56:13.000000000 +0400
@@ -1414,6 +1414,8 @@ endmenu
```

```
 source "lib/Kconfig"

+source "kernel/bc/Kconfig"
+
 source "arch/powerpc/oprofile/Kconfig"

 source "arch/ppc/Kconfig.debug"
--- ./arch/sparc/Kconfig.arkcfg 2006-04-21 11:59:32.000000000 +0400
+++ ./arch/sparc/Kconfig 2006-08-10 17:56:24.000000000 +0400
@@ -296,3 +296,5 @@ source "security/Kconfig"
 source "crypto/Kconfig"

 source "lib/Kconfig"
+
+source "kernel/bc/Kconfig"
--- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
+++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
@@ -432,3 +432,5 @@ source "security/Kconfig"
 source "crypto/Kconfig"

 source "lib/Kconfig"
+
+source "kernel/bc/Kconfig"
--- ./arch/x86_64/Kconfig.bckm 2006-07-10 12:39:11.000000000 +0400
+++ ./arch/x86_64/Kconfig 2006-07-28 14:10:49.000000000 +0400
@@ -655,3 +655,5 @@ source "security/Kconfig"
 source "crypto/Kconfig"

 source "lib/Kconfig"
+
+source "kernel/bc/Kconfig"
--- ./kernel/bc/Kconfig.bckm 2006-07-28 13:07:38.000000000 +0400
+++ ./kernel/bc/Kconfig 2006-07-28 13:09:51.000000000 +0400
@@ -0,0 +1,25 @@
+#
+# Resource beancounters (BC)
+#
+# Copyright (C) 2006 OpenVZ. SWsoft Inc
+
+menu "User resources"
+
+config BEANCOUNTERS
+ bool "Enable resource accounting/control"
+ default n
+ help
+         This patch provides accounting and allows to configure
+         limits for user's consumption of exhaustible system resources.
+         The most important resource controlled by this patch is unswappable
```

```
+	  memory (either mlock'ed or used by internal kernel structures and
+	  buffers). The main goal of this patch is to protect processes
+	  from running short of important resources because of an accidental
+	  misbehavior of processes or malicious activity aiming to ``kill''
+	  the system. It's worth to mention that resource limits configured
+	  by setrlimit(2) do not give an acceptable level of protection
+	  because they cover only small fraction of resources and work on a
+	  per-process basis.  Per-process accounting doesn't prevent malicious
+	  users from spawning a lot of resource-consuming processes.
+
+endmenu
```

Subject: [PATCH 2/6] BC: beancounters core (API)
Posted by dev on Wed, 23 Aug 2006 11:00:37 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 |  140 ++++++++++++++++++++++
 init/main.c              |    4 
 kernel/Makefile          |    1 
 kernel/bc/Makefile       |    7 +
 kernel/bc/beancounter.c  |  292 ++++++++++++++++++++++++++++++++++++++++++++++++
 5 files changed, 444 insertions(+)

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/beancounter.h 2006-08-21 13:14:01.000000000 +0400
@@ -0,0 +1,140 @@
+/*
+ *  include/bc/beancounter.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
```

```
+#ifndef _LINUX_BEANCOUNTER_H
+#define _LINUX_BEANCOUNTER_H
+
+/*
+ * Resource list.
+ */
+
+#define BC_RESOURCES 0
+
+struct bc_resource_parm {
+ unsigned long barrier; /* A barrier over which resource allocations
+     * are failed gracefully. e.g. if the amount
+     * of consumed memory is over the barrier
+     * further sbrk() or mmap() calls fail, the
+     * existing processes are not killed.
+     */
+ unsigned long limit; /* hard resource limit */
+ unsigned long held; /* consumed resources */
+ unsigned long maxheld; /* maximum amount of consumed resources */
+ unsigned long minheld; /* minumum amount of consumed resources */
+ unsigned long failcnt; /* count of failed charges */
+};
+
+/*
+ * Kernel internal part.
+ */
+
+#ifdef __KERNEL__
+
+#include <linux/config.h>
+#include <linux/spinlock.h>
+#include <linux/list.h>
+#include <asm/atomic.h>
+
+#define BC_MAXVALUE LONG_MAX
+
+/*
+ * Resource management structures
+ * Serialization issues:
+ *   beancounter list management is protected via bc_hash_lock
+ *   task pointers are set only for current task and only once
+ *   refcount is managed atomically
+ *   value and limit comparison and change are protected by per-bc spinlock
+ */
+
+struct beancounter
+{
+ atomic_t  bc_refcount;
```

```
+ spinlock_t  bc_lock;
+ uid_t   bc_id;
+ struct hlist_node hash;
+
+ /* resources statistics and settings */
+ struct bc_resource_parm bc_parms[BC_RESOURCES];
+};
+
+enum severity { BC_BARRIER, BC_LIMIT, BC_FORCE };
+
+/* Flags passed to beancounter_findcreate() */
+#define BC_ALLOC  0x01 /* May allocate new one */
+#define BC_ALLOC_ATOMIC  0x02 /* Allocate with GFP_ATOMIC */
+
+#define BC_HASH_SIZE  256
+
+#ifdef CONFIG_BEANCOUNTERS
+extern struct hlist_head bc_hash[];
+extern spinlock_t bc_hash_lock;
+
+/*
+ * This function tunes minheld and maxheld values for a given
+ * resource when held value changes
+ */
+static inline void bc_adjust_held_minmax(struct beancounter *bc,
+  int resource)
+{
+ if (bc->bc_parms[resource].maxheld < bc->bc_parms[resource].held)
+  bc->bc_parms[resource].maxheld = bc->bc_parms[resource].held;
+ if (bc->bc_parms[resource].minheld > bc->bc_parms[resource].held)
+  bc->bc_parms[resource].minheld = bc->bc_parms[resource].held;
+}
+
+void bc_print_resource_warning(struct beancounter *bc, int res,
+  char *str, unsigned long val, unsigned long held);
+void bc_print_id(struct beancounter *bc, char *str, int size);
+
+int bc_charge_locked(struct beancounter *bc,
+  int res, unsigned long val, enum severity strict);
+int bc_charge(struct beancounter *bc,
+  int res, unsigned long val, enum severity strict);
+
+void bc_uncharge_locked(struct beancounter *bc,
+  int res, unsigned long val);
+void bc_uncharge(struct beancounter *bc,
+  int res, unsigned long val);
+
+struct beancounter *beancounter_findcreate(uid_t id, int mask);
```

```
+
+static inline struct beancounter *get_beancounter(struct beancounter *bc)
+{
+ atomic_inc(&bc->bc_refcount);
+ return bc;
+}
+
+void __put_beancounter(struct beancounter *bc);
+static inline void put_beancounter(struct beancounter *bc)
+{
+ __put_beancounter(bc);
+}
+
+void bc_init_early(void);
+void bc_init_late(void);
+void bc_init_proc(void);
+
+extern struct beancounter init_bc;
+extern const char *bc_rnames[];
+
+#else /* CONFIG_BEANCOUNTERS */
+
+#define beancounter_findcreate(id, f)   (NULL)
+#define get_beancounter(bc)    (NULL)
+#define put_beancounter(bc)    do { } while (0)
+#define bc_charge_locked(bc, r, v, s)   (0)
+#define bc_charge(bc, r, v)    (0)
+#define bc_uncharge_locked(bc, r, v)   do { } while (0)
+#define bc_uncharge(bc, r, v)    do { } while (0)
+#define bc_init_early()     do { } while (0)
+#define bc_init_late()     do { } while (0)
+#define bc_init_proc()     do { } while (0)
+
+#endif /* CONFIG_BEANCOUNTERS */
+#endif /* __KERNEL__ */
+
+#endif /* _LINUX_BEANCOUNTER_H */
--- ./init/main.c.ve1 2006-08-21 12:25:13.000000000 +0400
+++ ./init/main.c 2006-08-21 12:45:32.000000000 +0400
@@ -52,6 +52,8 @@
 #include <linux/debug_locks.h>
 #include <linux/lockdep.h>

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

```
@@ -494,6 +496,7 @@ asmlinkage void __init start_kernel(void
  early_boot_irqs_off();
  early_init_irq_lock_class();

+ bc_init_early();
 /*
  * Interrupts are still disabled. Do necessary setups, then
  * enable them
@@ -587,6 +590,7 @@ asmlinkage void __init start_kernel(void
 #endif
  fork_init(num_physpages);
  proc_caches_init();
+ bc_init_late();
  buffer_init();
  unnamed_dev_init();
  key_init();
--- ./kernel/Makefile.ve1 2006-08-21 12:25:14.000000000 +0400
+++ ./kernel/Makefile 2006-08-21 12:25:27.000000000 +0400
@@ -12,6 +12,7 @@ obj-y     = sched.o fork.o exec_domain.o

 obj-$(CONFIG_STACKTRACE) += stacktrace.o
 obj-y += time/
+obj-y += bc/
 obj-$(CONFIG_DEBUG_MUTEXES) += mutex-debug.o
 obj-$(CONFIG_LOCKDEP) += lockdep.o
 ifeq ($(CONFIG_PROC_FS),y)
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/Makefile 2006-08-21 12:25:27.000000000 +0400
@@ -0,0 +1,7 @@
+#
+# Beancounters (BC)
+#
+# Copyright (C) 2006 OpenVZ. SWsoft Inc
+#
+
+obj-$(CONFIG_BEANCOUNTERS) += beancounter.o
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/beancounter.c 2006-08-21 13:13:11.000000000 +0400
@@ -0,0 +1,292 @@
+/*
+ *  kernel/bc/beancounter.c
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *  Original code by (C) 1998      Alan Cox
+ *                   1998-2000 Andrey Savochkin <saw@saw.sw.com.sg>
+ */
+
+#include <linux/slab.h>
```

```
+#include <linux/module.h>
+
+#include <bc/beancounter.h>
+
+static kmem_cache_t *bc_cachep;
+static struct beancounter default_beancounter;
+
+static void init_beancounter_struct(struct beancounter *bc, uid_t id);
+
+struct beancounter init_bc;
+
+const char *bc_rnames[] = {
+};
+
+#define bc_hash_fun(x) ((((x) >> 8) ^ (x)) & (BC_HASH_SIZE - 1))
+
+struct hlist_head bc_hash[BC_HASH_SIZE];
+spinlock_t bc_hash_lock;
+
+EXPORT_SYMBOL(bc_hash);
+EXPORT_SYMBOL(bc_hash_lock);
+
+/*
+ * Per resource beancounting. Resources are tied to their luid.
+ * The resource structure itself is tagged both to the process and
+ * the charging resources (a socket doesn't want to have to search for
+ * things at irq time for example). Reference counters keep things in
+ * hand.
+ *
+ * The case where a user creates resource, kills all his processes and
+ * then starts new ones is correctly handled this way. The refcounters
+ * will mean the old entry is still around with resource tied to it.
+ */
+
+struct beancounter *beancounter_findcreate(uid_t uid, int mask)
+{
+ struct beancounter *new_bc, *bc;
+ unsigned long flags;
+ struct hlist_head *slot;
+ struct hlist_node *pos;
+
+ slot = &bc_hash[bc_hash_fun(uid)];
+ new_bc = NULL;
+
+retry:
+ spin_lock_irqsave(&bc_hash_lock, flags);
+ hlist_for_each_entry (bc, pos, slot, hash)
+  if (bc->bc_id == uid)
```

```
+   break;
+
+ if (pos != NULL) {
+  get_beancounter(bc);
+  spin_unlock_irqrestore(&bc_hash_lock, flags);
+
+  if (new_bc != NULL)
+   kmem_cache_free(bc_cachep, new_bc);
+  return bc;
+ }
+
+ if (!(mask & BC_ALLOC))
+  goto out_unlock;
+
+ if (new_bc != NULL)
+  goto out_install;
+
+ spin_unlock_irqrestore(&bc_hash_lock, flags);
+
+ new_bc = kmem_cache_alloc(bc_cachep,
+   mask & BC_ALLOC_ATOMIC ? GFP_ATOMIC : GFP_KERNEL);
+ if (new_bc == NULL)
+  goto out;
+
+ memcpy(new_bc, &default_beancounter, sizeof(*new_bc));
+ init_beancounter_struct(new_bc, uid);
+ goto retry;
+
+out_install:
+ hlist_add_head(&new_bc->hash, slot);
+out_unlock:
+ spin_unlock_irqrestore(&bc_hash_lock, flags);
+out:
+ return new_bc;
+}
+
+void bc_print_id(struct beancounter *bc, char *str, int size)
+{
+ snprintf(str, size, "%u", bc->bc_id);
+}
+
+void bc_print_resource_warning(struct beancounter *bc, int res,
+  char *str, unsigned long val, unsigned long held)
+{
+ char uid[64];
+
+ bc_print_id(bc, uid, sizeof(uid));
+ printk(KERN_WARNING "BC %s %s warning: %s "
```

```
+   "(held %lu, fails %lu, val %lu)\n",
+   uid, bc_rnames[res], str,
+   (res < BC_RESOURCES ? bc->bc_parms[res].held : held),
+   (res < BC_RESOURCES ? bc->bc_parms[res].failcnt : 0),
+   val);
+}
+
+static inline void verify_held(struct beancounter *bc)
+{
+ int i;
+
+ for (i = 0; i < BC_RESOURCES; i++)
+  if (bc->bc_parms[i].held != 0)
+   bc_print_resource_warning(bc, i,
+     "resource is held on put", 0, 0);
+}
+
+void __put_beancounter(struct beancounter *bc)
+{
+ unsigned long flags;
+
+ /* equivalent to atomic_dec_and_lock_irqsave() */
+ local_irq_save(flags);
+ if (likely(!atomic_dec_and_lock(&bc->bc_refcount, &bc_hash_lock))) {
+  local_irq_restore(flags);
+  if (unlikely(atomic_read(&bc->bc_refcount) < 0))
+   printk(KERN_ERR "BC: Bad refcount: bc=%p, "
+     "luid=%d, ref=%d\n",
+     bc, bc->bc_id,
+     atomic_read(&bc->bc_refcount));
+  return;
+ }
+
+ BUG_ON(bc == &init_bc);
+ verify_held(bc);
+ hlist_del(&bc->hash);
+ spin_unlock_irqrestore(&bc_hash_lock, flags);
+ kmem_cache_free(bc_cachep, bc);
+}
+
+EXPORT_SYMBOL(__put_beancounter);
+
+/*
+ * Generic resource charging stuff
+ */
+
+/* called with bc->bc_lock held and interrupts disabled */
+int bc_charge_locked(struct beancounter *bc, int resource, unsigned long val,
```

```
+  enum severity strict)
+{
+ /*
+  * bc_value <= BC_MAXVALUE, value <= BC_MAXVALUE, and only one addition
+  * at the moment is possible so an overflow is impossible.
+  */
+ bc->bc_parms[resource].held += val;
+
+ switch (strict) {
+  case BC_BARRIER:
+   if (bc->bc_parms[resource].held >
+     bc->bc_parms[resource].barrier)
+    break;
+   /* fallthrough */
+  case BC_LIMIT:
+   if (bc->bc_parms[resource].held >
+     bc->bc_parms[resource].limit)
+    break;
+   /* fallthrough */
+  case BC_FORCE:
+   bc_adjust_held_minmax(bc, resource);
+   return 0;
+  default:
+   BUG();
+ }
+
+ bc->bc_parms[resource].failcnt++;
+ bc->bc_parms[resource].held -= val;
+ return -ENOMEM;
+}
+EXPORT_SYMBOL(bc_charge_locked);
+
+int bc_charge(struct beancounter *bc, int resource, unsigned long val,
+  enum severity strict)
+{
+ int retval;
+ unsigned long flags;
+
+ BUG_ON(val > BC_MAXVALUE);
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ retval = bc_charge_locked(bc, resource, val, strict);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+ return retval;
+}
+EXPORT_SYMBOL(bc_charge);
+
+/* called with bc->bc_lock held and interrupts disabled */
```

```
+void bc_uncharge_locked(struct beancounter *bc, int resource, unsigned long val)
+{
+ if (unlikely(bc->bc_parms[resource].held < val)) {
+  bc_print_resource_warning(bc, resource,
+    "uncharging too much", val, 0);
+  val = bc->bc_parms[resource].held;
+ }
+
+ bc->bc_parms[resource].held -= val;
+ bc_adjust_held_minmax(bc, resource);
+}
+EXPORT_SYMBOL(bc_uncharge_locked);
+
+void bc_uncharge(struct beancounter *bc, int resource, unsigned long val)
+{
+ unsigned long flags;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ bc_uncharge_locked(bc, resource, val);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+}
+EXPORT_SYMBOL(bc_uncharge);
+
+/*
+ * Initialization
+ *
+ * struct beancounter contains
+ *  - limits and other configuration settings
+ *  - structural fields: lists, spinlocks and so on.
+ *
+ * Before these parts are initialized, the structure should be memset
+ * to 0 or copied from a known clean structure.  That takes care of a lot
+ * of fields not initialized explicitly.
+ */
+
+static void init_beancounter_struct(struct beancounter *bc, uid_t id)
+{
+ atomic_set(&bc->bc_refcount, 1);
+ spin_lock_init(&bc->bc_lock);
+ bc->bc_id = id;
+}
+
+static void init_beancounter_nolimits(struct beancounter *bc)
+{
+ int k;
+
+ for (k = 0; k < BC_RESOURCES; k++) {
+  bc->bc_parms[k].limit = BC_MAXVALUE;
```

```
+  bc->bc_parms[k].barrier = BC_MAXVALUE;
+ }
+}
+
+static void init_beancounter_syslimits(struct beancounter *bc)
+{
+ int k;
+
+ for (k = 0; k < BC_RESOURCES; k++)
+  bc->bc_parms[k].barrier = bc->bc_parms[k].limit;
+}
+
+void __init bc_init_early(void)
+{
+ struct beancounter *bc;
+ struct hlist_head *slot;
+
+ bc = &init_bc;
+
+ memset(bc, 0, sizeof(*bc));
+ init_beancounter_nolimits(bc);
+ init_beancounter_struct(bc, 0);
+
+ spin_lock_init(&bc_hash_lock);
+ slot = &bc_hash[bc_hash_fun(bc->bc_id)];
+ hlist_add_head(&bc->hash, slot);
+}
+
+void __init bc_init_late(void)
+{
+ struct beancounter *bc;
+
+ bc_cachep = kmem_cache_create("beancounters",
+   sizeof(struct beancounter),
+   0, SLAB_HWCACHE_ALIGN, NULL, NULL);
+ if (bc_cachep == NULL)
+  panic("Can't create bc caches\n");
+
+ bc = &default_beancounter;
+ memset(bc, 0, sizeof(default_beancounter));
+ init_beancounter_syslimits(bc);
+ init_beancounter_struct(bc, 0);
+}
```

Subject: [PATCH 3/6] BC: context inheriting and changing
Posted by dev on Wed, 23 Aug 2006 11:02:40 GMT

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

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

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

```
 include/linux/sched.h   |    5 ++++
 include/bc/task.h       |   52 +++++++++++++++++++++++++++++++++++++++++++++++++
 kernel/fork.c           |   17 ++++++++++++--
 kernel/irq/handle.c     |    9 ++++++++
 kernel/softirq.c        |    8 +++++++
 kernel/bc/Makefile      |    1
 kernel/bc/beancounter.c |    3 ++
 kernel/bc/misc.c        |   32 ++++++++++++++++++++++++++++
 8 files changed, 125 insertions(+), 2 deletions(-)

--- ./include/linux/sched.h.ve2 2006-08-21 13:15:39.000000000 +0400
+++ ./include/linux/sched.h 2006-08-21 13:26:03.000000000 +0400
@@ -83,6 +83,8 @@ struct sched_param {
 #include <linux/timer.h>
 #include <linux/hrtimer.h>

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

 struct exec_domain;
@@ -1032,6 +1034,9 @@ struct task_struct {
  spinlock_t delays_lock;
  struct task_delay_info *delays;
 #endif
+#ifdef CONFIG_BEANCOUNTERS
+ struct task_beancounter task_bc;
+#endif
 };

 static inline pid_t process_group(struct task_struct *tsk)
--- ./include/bc/task.h.ve2 2006-08-21 13:26:03.000000000 +0400
+++ ./include/bc/task.h 2006-08-21 13:35:04.000000000 +0400
```

```
@@ -0,0 +1,52 @@
+/*
+ *  include/bc/task.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#ifndef __BC_TASK_H_
+#define __BC_TASK_H_
+
+#include <linux/config.h>
+
+struct beancounter;
+
+struct task_beancounter {
+ struct beancounter *exec_bc;
+ struct beancounter *fork_bc;
+};
+
+#ifdef CONFIG_BEANCOUNTERS
+
+#define get_exec_bc()  (current->task_bc.exec_bc)
+#define set_exec_bc(new)   \
+ ({     \
+   struct beancounter *old; \
+   struct task_beancounter *tbc; \
+   tbc = &current->task_bc; \
+   old = tbc->exec_bc;  \
+   tbc->exec_bc = new;  \
+   old;    \
+  })
+#define reset_exec_bc(old, exp)   \
+ do {     \
+  struct task_beancounter *tbc; \
+  tbc = &current->task_bc; \
+  BUG_ON(tbc->exec_bc != exp); \
+  tbc->exec_bc = old;  \
+ } while (0)
+
+int bc_task_charge(struct task_struct *parent, struct task_struct *new);
+void bc_task_uncharge(struct task_struct *tsk);
+
+#else
+
+#define get_exec_bc()  (NULL)
+#define set_exec_bc(new) (NULL)
+#define reset_exec_bc(new, exp) do { } while (0)
```

```
+#define bc_task_charge(p, t) (0)
+#define bc_task_uncharge(p) do { } while (0)
+
+#endif
+#endif
--- ./kernel/fork.c.ve2 2006-08-21 13:15:39.000000000 +0400
+++ ./kernel/fork.c 2006-08-21 13:40:16.000000000 +0400
@@ -48,6 +48,8 @@
 #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>
@@ -102,12 +104,18 @@ kmem_cache_t *vm_area_cachep;
 /* SLAB cache for mm_struct structures (tsk->mm) */
 static kmem_cache_t *mm_cachep;

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

 void __put_task_struct(struct task_struct *tsk)
@@ -978,6 +986,9 @@ static struct task_struct *copy_process(
  if (!p)
   goto fork_out;

+ if (bc_task_charge(current, p))
+  goto bad_charge;
+
 #ifdef CONFIG_TRACE_IRQFLAGS
  DEBUG_LOCKS_WARN_ON(!p->hardirqs_enabled);
  DEBUG_LOCKS_WARN_ON(!p->softirqs_enabled);
@@ -1291,7 +1302,9 @@ bad_fork_cleanup_count:
  atomic_dec(&p->user->processes);
```

```
  free_uid(p->user);
 bad_fork_free:
- free_task(p);
+ bc_task_uncharge(p);
+bad_charge:
+ __free_task(p);
 fork_out:
  return ERR_PTR(retval);
 }
--- ./kernel/irq/handle.c.ve2 2006-07-10 12:39:20.000000000 +0400
+++ ./kernel/irq/handle.c 2006-08-21 13:42:13.000000000 +0400
@@ -16,6 +16,9 @@
 #include <linux/interrupt.h>
 #include <linux/kernel_stat.h>

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

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

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

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

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

```
--- ./kernel/softirq.c.ve2 2006-08-21 13:15:39.000000000 +0400
+++ ./kernel/softirq.c 2006-08-21 13:41:04.000000000 +0400
@@ -18,6 +18,9 @@
 #include <linux/rcupdate.h>
 #include <linux/smp.h>

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

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

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

 #ifndef __ARCH_HAS_DO_SOFTIRQ
--- ./kernel/bc/Makefile.ve2 2006-08-21 13:26:03.000000000 +0400
+++ ./kernel/bc/Makefile 2006-08-21 13:26:03.000000000 +0400
@@ -5,3 +5,4 @@
 #

 obj-$(CONFIG_BEANCOUNTERS) += beancounter.o
+obj-$(CONFIG_BEANCOUNTERS) += misc.o
--- ./kernel/bc/beancounter.c.ve2 2006-08-21 13:13:11.000000000 +0400
+++ ./kernel/bc/beancounter.c 2006-08-21 13:13:11.000000000 +0400
@@ -280,6 +280,9 @@
  spin_lock_init(&bc_hash_lock);
  slot = &bc_hash[bc_hash_fun(bc->bc_id)];
  hlist_add_head(&bc->hash, slot);
+
+ current->task_bc.exec_bc = get_beancounter(bc);
+ current->task_bc.fork_bc = get_beancounter(bc);
 }
```

```
 void __init bc_init_late(void)
--- ./kernel/bc/misc.c.ve2 2006-08-21 13:26:03.000000000 +0400
+++ ./kernel/bc/misc.c 2006-08-21 13:39:07.000000000 +0400
@@ -0,0 +1,32 @@
+/*
+ * kernel/bc/misc.c
+ *
+ * Copyright (C) 2006 OpenVZ. SWsoft Inc.
+ *
+ */
+
+#include <linux/sched.h>
+
+#include <bc/beancounter.h>
+#include <bc/task.h>
+
+int bc_task_charge(struct task_struct *parent, struct task_struct *new)
+{
+ struct task_beancounter *old_bc;
+ struct task_beancounter *new_bc;
+ struct beancounter *bc;
+
+ old_bc = &parent->task_bc;
+ new_bc = &new->task_bc;
+
+ bc = old_bc->fork_bc;
+ new_bc->exec_bc = get_beancounter(bc);
+ new_bc->fork_bc = get_beancounter(bc);
+ return 0;
+}
+
+void bc_task_uncharge(struct task_struct *tsk)
+{
+ put_beancounter(tsk->task_bc.exec_bc);
+ put_beancounter(tsk->task_bc.fork_bc);
+}
```

Subject: [PATCH 4/6] BC: user interface (syscalls)
Posted by dev on Wed, 23 Aug 2006 11:03:39 GMT
View Forum Message <> Reply to Message

Add the following system calls for BC management:

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

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

 arch/i386/kernel/syscall_table.S |    4 +
 arch/ia64/kernel/entry.S         |    4 +
 arch/sparc/kernel/entry.S        |    2
 arch/sparc/kernel/systbls.S      |    6 +
 arch/sparc64/kernel/entry.S      |    2
 arch/sparc64/kernel/systbls.S    |   10 ++-
 include/asm-i386/unistd.h        |    6 +
 include/asm-ia64/unistd.h        |    6 +
 include/asm-powerpc/systbl.h     |    4 +
 include/asm-powerpc/unistd.h     |    6 +
 include/asm-sparc/unistd.h       |    4 +
 include/asm-sparc64/unistd.h     |    4 +
 include/asm-x86_64/unistd.h      |   10 ++-
 kernel/sys_ni.c                  |    6 +
 kernel/bc/Makefile               |    1
 kernel/bc/sys.c                  |  120 ++++++++++++++++++++++++++++++++++++++++
 16 files changed, 186 insertions(+), 9 deletions(-)
--- ./arch/i386/kernel/syscall_table.S.ve3 2006-08-21 13:15:37.000000000 +0400
+++ ./arch/i386/kernel/syscall_table.S 2006-08-21 14:15:47.000000000 +0400
@@ -318,3 +318,7 @@ ENTRY(sys_call_table)
  .long sys_vmsplice
  .long sys_move_pages
  .long sys_getcpu
+ .long sys_get_bcid
+ .long sys_set_bcid  /* 320 */
+ .long sys_set_bclimit
+ .long sys_get_bcstat
--- ./arch/ia64/kernel/entry.S.ve3 2006-08-21 13:15:37.000000000 +0400
+++ ./arch/ia64/kernel/entry.S 2006-08-21 14:17:07.000000000 +0400
@@ -1610,5 +1610,9 @@ sys_call_table:
  data8 sys_sync_file_range  // 1300
  data8 sys_tee
  data8 sys_vmsplice
+ data8 sys_get_bcid
+ data8 sys_set_bcid
+ data8 sys_set_bclimit   // 1305
+ data8 sys_get_bcstat

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

```
 #define curptr      g6

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

 /* These are just handy. */
 #define _SV save %sp, -STACKFRAME_SZ, %sp
--- ./arch/sparc/kernel/systbls.S.ve3 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/sparc/kernel/systbls.S 2006-08-21 14:30:43.000000000 +0400
@@ -78,7 +78,8 @@ sys_call_table:
 /*285*/ .long sys_mkdirat, sys_mknodat, sys_fchownat, sys_futimesat, sys_fstatat64
 /*290*/ .long sys_unlinkat, sys_renameat, sys_linkat, sys_symlinkat, sys_readlinkat
 /*295*/ .long sys_fchmodat, sys_faccessat, sys_pselect6, sys_ppoll, sys_unshare
-/*300*/ .long sys_set_robust_list, sys_get_robust_list
+/*300*/ .long sys_set_robust_list, sys_get_robust_list, sys_get_bcid, sys_set_bcid,
sys_set_bclimit
+/*305*/ .long sys_get_bcstat

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

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

 #define curptr      g6

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

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

```
+ .word sys_nis_syscall

 #endif /* CONFIG_COMPAT */

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

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

 #ifdef __KERNEL__

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

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

```
+#define __NR_get_bcid   1303
+#define __NR_set_bcid   1304
+#define __NR_set_bclimit  1305
+#define __NR_get_bcstat   1306

 #ifdef __KERNEL__


-#define NR_syscalls   279 /* length of syscall table */
+#define NR_syscalls   283 /* length of syscall table */

 #define __ARCH_WANT_SYS_RT_SIGACTION

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

 #ifdef __KERNEL__

-#define __NR_syscalls  301
+#define __NR_syscalls  305

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

```
+#define __NR_set_bcid  303
+#define __NR_set_bclimit 304
+#define __NR_get_bcstat  305

 #ifdef __KERNEL__
 /* WARNING: You MAY NOT add syscall numbers larger than 301, since
--- ./include/asm-sparc64/unistd.h.ve3 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-sparc64/unistd.h 2006-08-21 14:34:10.000000000 +0400
@@ -320,6 +320,10 @@
 #define __NR_unshare  299
 #define __NR_set_robust_list 300
 #define __NR_get_robust_list 301
+#define __NR_get_bcid  302
+#define __NR_set_bcid  303
+#define __NR_set_bclimit 304
+#define __NR_get_bcstat  305

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

 #ifdef __KERNEL__

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

 #ifndef __NO_STUBS
--- ./kernel/sys_ni.c.ve3 2006-07-10 12:39:20.000000000 +0400
+++ ./kernel/sys_ni.c 2006-08-21 14:12:49.000000000 +0400
@@ -134,3 +134,9 @@ cond_syscall(sys_madvise);
 cond_syscall(sys_mremap);
 cond_syscall(sys_remap_file_pages);
 cond_syscall(compat_sys_move_pages);
+
```

```
+/* user resources syscalls */
+cond_syscall(sys_set_bcid);
+cond_syscall(sys_get_bcid);
+cond_syscall(sys_set_bclimit);
+cond_syscall(sys_get_bcstat);
--- ./kernel/bc/Makefile.ve3 2006-08-21 13:49:49.000000000 +0400
+++ ./kernel/bc/Makefile 2006-08-21 13:55:39.000000000 +0400
@@ -6,3 +6,4 @@

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

```
+  goto out;
+
+ /* Ok - set up a beancounter entry for this user */
+ error = -ENOMEM;
+ bc = beancounter_findcreate(id, BC_ALLOC);
+ if (bc == NULL)
+  goto out;
+
+ /* install bc */
+ put_beancounter(task_bc->exec_bc);
+ task_bc->exec_bc = bc;
+ put_beancounter(task_bc->fork_bc);
+ task_bc->fork_bc = get_beancounter(bc);
+ error = 0;
+out:
+ return error;
+}
+
+asmlinkage long sys_set_bclimit(uid_t id, unsigned long resource,
+  unsigned long *limits)
+{
+ int error;
+ unsigned long flags;
+ struct beancounter *bc;
+ unsigned long new_limits[2];
+
+ error = -EPERM;
+ if(!capable(CAP_SYS_RESOURCE))
+  goto out;
+
+ error = -EINVAL;
+ if (resource >= BC_RESOURCES)
+  goto out;
+
+ error = -EFAULT;
+ if (copy_from_user(&new_limits, limits, sizeof(new_limits)))
+  goto out;
+
+ error = -EINVAL;
+ if (new_limits[0] > BC_MAXVALUE || new_limits[1] > BC_MAXVALUE)
+  goto out;
+
+ error = -ENOENT;
+ bc = beancounter_findcreate(id, 0);
+ if (bc == NULL)
+  goto out;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
```

```
+ bc->bc_parms[resource].barrier = new_limits[0];
+ bc->bc_parms[resource].limit = new_limits[1];
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+
+ put_beancounter(bc);
+ error = 0;
+out:
+ return error;
+}
+
+int sys_get_bcstat(uid_t id, unsigned long resource,
+  struct bc_resource_parm *uparm)
+{
+ int error;
+ unsigned long flags;
+ struct beancounter *bc;
+ struct bc_resource_parm parm;
+
+ error = -EINVAL;
+ if (resource > BC_RESOURCES)
+  goto out;
+
+ error = -ENOENT;
+ bc = beancounter_findcreate(id, 0);
+ if (bc == NULL)
+  goto out;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ parm = bc->bc_parms[resource];
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+ put_beancounter(bc);
+
+ error = 0;
+ if (copy_to_user(uparm, &parm, sizeof(parm)))
+  error = -EFAULT;
+
+out:
+ return error;
+}
```

Subject: [PATCH 5/6] BC: kernel memory accounting (core)
Posted by dev on Wed, 23 Aug 2006 11:04:41 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/linux/gfp.h | 8 ++-
 include/linux/mm.h | 6 ++
 include/linux/slab.h | 4 +
 include/linux/vmalloc.h | 1
 include/bc/beancounter.h | 4 +
 include/bc/kmem.h | 33 ++++++++++++
 kernel/bc/Makefile | 1
 kernel/bc/beancounter.c | 3 +
 kernel/bc/kmem.c | 89 ++++++++++++++++++++++++++++++++++++
 mm/mempool.c | 2
 mm/page_alloc.c | 11 ++++
 mm/slab.c | 121 +++----------
 mm/vmalloc.c | 6 ++
 13 files changed, 264 insertions(+), 25 deletions(-)

--- ./include/linux/gfp.h.bckmem 2006-07-28 18:43:52.000000000 +0400
+++ ./include/linux/gfp.h 2006-07-31 16:32:22.000000000 +0400
@@ -46,8 +46,10 @@ 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 */
@@ -54,7 +56,8 @@ struct vm_area_struct;

```
 #define GFP_LEVEL_MASK (__GFP_WAIT|__GFP_HIGH|__GFP_IO|__GFP_FS| \
    __GFP_COLD|__GFP_NOWARN|__GFP_REPEAT| \
    __GFP_NOFAIL|__GFP_NORETRY|__GFP_NO_GROW|__GFP_COMP| \
-   __GFP_NOMEMALLOC|__GFP_HARDWALL|__GFP_THISNODE)
+   __GFP_NOMEMALLOC|__GFP_HARDWALL|__GFP_THISNODE| \
+   __GFP_BC|__GFP_BC_LIMIT)

 /* This equals 0, but use constants in case they ever change */
 #define GFP_NOWAIT (GFP_ATOMIC & ~__GFP_HIGH)
@@ -63,6 +66,7 @@ struct vm_area_struct;
 #define GFP_NOIO (__GFP_WAIT)
 #define GFP_NOFS (__GFP_WAIT | __GFP_IO)
 #define GFP_KERNEL (__GFP_WAIT | __GFP_IO | __GFP_FS)
+#define GFP_KERNEL_BC (__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_BC)
 #define GFP_USER (__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_HARDWALL)
 #define GFP_HIGHUSER (__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_HARDWALL | \
     __GFP_HIGHMEM)
--- ./include/linux/mm.h.bckmem 2006-07-28 18:43:52.000000000 +0400
+++ ./include/linux/mm.h 2006-07-31 17:50:29.000000000 +0400
@@ -267,8 +267,14 @@ struct page {
  unsigned int gfp_mask;
  unsigned long trace[8];
 #endif
+#ifdef CONFIG_BEANCOUNTERS
+ union {
+  struct beancounter *page_bc;
+ } bc;
+#endif
 };

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

--- ./include/linux/slab.h.bckmem 2006-07-10 12:39:19.000000000 +0400
+++ ./include/linux/slab.h 2006-07-31 17:02:08.000000000 +0400
@@ -46,6 +46,8 @@ typedef struct kmem_cache kmem_cache_t;
 #define SLAB_PANIC  0x00040000UL /* panic if kmem_cache_create() fails */
 #define SLAB_DESTROY_BY_RCU 0x00080000UL /* defer freeing pages to RCU */
 #define SLAB_MEM_SPREAD  0x00100000UL /* Spread some memory over cpuset */
+#define SLAB_BC  0x00200000UL /* Account with BC */
+#define SLAB_BC_NOCHARGE 0x00400000UL /* Explicit accounting */

 /* flags passed to a constructor func */
 #define SLAB_CTOR_CONSTRUCTOR 0x001UL  /* if not set, then deconstructor */
@@ -265,6 +267,8 @@ extern kmem_cache_t *bio_cachep;

 extern atomic_t slab_reclaim_pages;
```

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

 #endif /* _LINUX_SLAB_H */
--- ./include/linux/vmalloc.h.bckmem 2006-07-17 17:01:12.000000000 +0400
+++ ./include/linux/vmalloc.h 2006-08-01 13:21:59.000000000 +0400
@@ -36,6 +36,7 @@ struct vm_struct {
  * Highlevel APIs for driver use
  */
 extern void *vmalloc(unsigned long size);
+extern void *vmalloc_bc(unsigned long size);
 extern void *vmalloc_user(unsigned long size);
 extern void *vmalloc_node(unsigned long size, int node);
 extern void *vmalloc_exec(unsigned long size);
--- ./include/bc/beancounter.h.bckmem 2006-07-28 18:43:52.000000000 +0400
+++ ./include/bc/beancounter.h 2006-08-03 16:03:01.000000000 +0400
@@ -14,7 +14,9 @@
  * Resource list.
  */

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

 struct resource_parm {
  /*
--- ./include/bc/kmem.h.bckmem 2006-07-28 18:43:52.000000000 +0400
+++ ./include/bc/kmem.h 2006-07-31 17:37:05.000000000 +0400
@@ -0,0 +1,33 @@
+/*
+ *  include/bc/kmem.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#ifndef __BC_KMEM_H_
+#define __BC_KMEM_H_
+
+#include <linux/config.h>
+
+/*
+ * BC_KMEMSIZE accounting
+ */
+
```

```
+struct mm_struct;
+struct page;
+struct beancounter;
+
+#ifdef CONFIG_BEANCOUNTERS
+int  bc_page_charge(struct page *page, int order, gfp_t flags);
+void bc_page_uncharge(struct page *page, int order);
+
+int bc_slab_charge(kmem_cache_t *cachep, void *obj, gfp_t flags);
+void bc_slab_uncharge(kmem_cache_t *cachep, void *obj);
+#else
+#define bc_page_charge(pg, o, mask) (0)
+#define bc_page_uncharge(pg, o)  do { } while (0)
+#define bc_slab_charge(cachep, o, f) (0)
+#define bc_slab_uncharge(cachep, o) do { } while (0)
+#endif
+#endif /* __BC_SLAB_H_ */
--- ./kernel/bc/Makefile.bcsys 2006-07-28 14:08:37.000000000 +0400
+++ ./kernel/bc/Makefile 2006-08-01 11:08:39.000000000 +0400
@@ -7,3 +7,4 @@
 obj-$(CONFIG_BEANCOUNTERS) += beancounter.o
 obj-$(CONFIG_BEANCOUNTERS) += misc.o
 obj-$(CONFIG_BEANCOUNTERS) += sys.o
+obj-$(CONFIG_BEANCOUNTERS) += kmem.o
--- ./kernel/bc/beancounter.c.bckmem 2006-07-28 18:43:52.000000000 +0400
+++ ./kernel/bc/beancounter.c 2006-08-03 16:14:17.000000000 +0400
@@ -19,6 +19,7 @@ static void init_beancounter_struct(stru
 struct beancounter init_bc;

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

 #define bc_hash_fun(x) ((((x) >> 8) ^ (x)) & (BC_HASH_SIZE - 1))
@@ -348,6 +378,8 @@ static void init_beancounter_syslimits(s
 {
  int k;

+ bc->bc_parms[BC_KMEMSIZE].limit = 32 * 1024 * 1024;
+
  for (k = 0; k < BC_RESOURCES; k++)
   bc->bc_parms[k].barrier = bc->bc_parms[k].limit;
 }
--- ./kernel/bc/kmem.c.bckmem 2006-07-31 16:32:22.000000000 +0400
+++ ./kernel/bc/kmem.c 2006-07-31 17:51:27.000000000 +0400
@@ -0,0 +1,89 @@
+/*
+ *  kernel/bc/kmem.c
```

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

```
+ bc_uncharge(bc, BC_KMEMSIZE, size);
+ put_beancounter(bc);
+ *slab_bcp = NULL;
+}
+
+/*
+ * Pages accounting
+ */
+
+int bc_page_charge(struct page *page, int order, gfp_t flags)
+{
+ struct beancounter *bc;
+
+ BUG_ON(page_bc(page) != NULL);
+
+ bc = get_exec_bc();
+ if (bc == NULL)
+  return 0;
+
+ if (bc_charge(bc, BC_KMEMSIZE, PAGE_SIZE << order,
+   (flags & __GFP_BC_LIMIT ? BC_LIMIT : BC_BARRIER)))
+  return -ENOMEM;
+
+ page_bc(page) = get_beancounter(bc);
+ return 0;
+}
+
+void bc_page_uncharge(struct page *page, int order)
+{
+ struct beancounter *bc;
+
+ bc = page_bc(page);
+ if (bc == NULL)
+  return;
+
+ bc_uncharge(bc, BC_KMEMSIZE, PAGE_SIZE << order);
+ put_beancounter(bc);
+ page_bc(page) = NULL;
+}
--- ./mm/mempool.c.bckmem 2006-04-21 11:59:36.000000000 +0400
+++ ./mm/mempool.c 2006-08-01 13:25:26.000000000 +0400
@@ -119,6 +119,7 @@ int mempool_resize(mempool_t *pool, int
  unsigned long flags;

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

  spin_lock_irqsave(&pool->lock, flags);
```

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

  gfp_temp = gfp_mask & ~(__GFP_WAIT|__GFP_IO);

--- ./mm/page_alloc.c.bckmem 2006-07-28 18:43:52.000000000 +0400
+++ ./mm/page_alloc.c 2006-07-31 19:52:08.000000000 +0400
@@ -38,6 +38,8 @@
 #include <linux/mempolicy.h>
 #include <linux/stop_machine.h>

+#include <bc/kmem.h>
+
 #include <asm/tlbflush.h>
 #include <asm/div64.h>
 #include "internal.h"
@@ -453,6 +455,8 @@ static void __free_pages_ok(struct page
  if (reserved)
   return;

+ bc_page_uncharge(page, order);
+
  kernel_map_pages(page, 1 << order, 0);
  local_irq_save(flags);
  __count_vm_events(PGFREE, 1 << order);
@@ -724,6 +728,8 @@ static void fastcall free_hot_cold_page(
  if (free_pages_check(page))
   return;

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

  pcp = &zone_pcp(zone, get_cpu())->pcp[cold];
@@ -1056,6 +1062,11 @@ 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 2006-07-17 17:01:12.000000000 +0400
+++ ./mm/slab.c 2006-08-01 13:24:06.000000000 +0400
@@ -109,6 +109,8 @@
 #include <linux/mutex.h>
 #include <linux/rtmutex.h>

+#include <bc/kmem.h>
+
 #include <asm/uaccess.h>
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>
@@ -176,11 +178,13 @@
     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

@@ -774,9 +778,33 @@ struct kmem_cache *kmem_find_general_cac
  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);
 }

 /*
@@ -821,20 +849,21 @@ 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;
@@ -1393,7 +1422,8 @@ 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) {
@@ -1401,7 +1431,8 @@ 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);
   }

@@ -1419,7 +1450,8 @@ 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);
    }

@@ -1869,7 +1901,8 @@ 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;
@@ -2170,8 +2203,8 @@ 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
@@ -2182,11 +2215,9 @@ 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. */
@@ -2435,6 +2466,30 @@ int kmem_cache_destroy(struct kmem_cache
 }
 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
@@ -2445,7 +2500,8 @@ 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 {
@@ -2456,14 +2512,14 @@ 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)
 {
@@ -2641,7 +2697,7 @@ 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;

@@ -2989,6 +3045,19 @@ 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)
 {
@@ -3002,6 +3071,12 @@ 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;
 }
@@ -3193,6 +3266,8 @@ static inline void __cache_free(struct k
  struct array_cache *ac = cpu_cache_get(cachep);

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

  if (cache_free_alien(cachep, objp))
--- ./mm/vmalloc.c.bckmem 2006-07-17 17:01:12.000000000 +0400
+++ ./mm/vmalloc.c 2006-08-01 15:27:24.000000000 +0400
@@ -518,6 +518,12 @@ void *vmalloc(unsigned long size)
 }
 EXPORT_SYMBOL(vmalloc);

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

Subject: [PATCH 6/6] BC: kernel memory accounting (marks)
Posted by dev on Wed, 23 Aug 2006 11:05:46 GMT

View Forum Message <> Reply to Message

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

kernel resources.

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

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

--- ./arch/i386/kernel/ldt.c.bcslabs 2006-04-21 11:59:31.000000000 +0400
+++ ./arch/i386/kernel/ldt.c 2006-08-01 13:22:30.000000000 +0400
@@ -39,9 +39,9 @@ static int alloc_ldt(mm_context_t *pc, i
  oldsize = pc->size;
  mincount = (mincount+511)&(~511);
  if (mincount*LDT_ENTRY_SIZE > PAGE_SIZE)
-  newldt = vmalloc(mincount*LDT_ENTRY_SIZE);
+  newldt = vmalloc_bc(mincount*LDT_ENTRY_SIZE);
  else
-  newldt = kmalloc(mincount*LDT_ENTRY_SIZE, GFP_KERNEL);
+  newldt = kmalloc(mincount*LDT_ENTRY_SIZE, GFP_KERNEL_BC);

  if (!newldt)
   return -ENOMEM;
--- ./arch/i386/mm/init.c.bcslabs 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/i386/mm/init.c 2006-08-01 13:17:07.000000000 +0400
@@ -680,7 +680,7 @@ 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)
@@ -689,7 +689,7 @@ 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.bcslabs 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/i386/mm/pgtable.c 2006-08-01 13:27:35.000000000 +0400
@@ -158,9 +158,11 @@ 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.bcslabs 2006-07-10 12:39:11.000000000 +0400
+++ ./drivers/char/tty_io.c 2006-08-01 15:21:21.000000000 +0400
@@ -158,7 +158,7 @@ static struct tty_struct *alloc_tty_stru

 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 *);
@@ -1495,7 +1495,7 @@ static int init_dev(struct tty_driver *d

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

```
+      GFP_KERNEL_BC);
   if (!tp)
    goto free_mem_out;
   *tp = driver->init_termios;
@@ -1503,7 +1503,7 @@ static int init_dev(struct tty_driver *d

  if (!*ltp_loc) {
   ltp = (struct termios *) kmalloc(sizeof(struct termios),
-       GFP_KERNEL);
+       GFP_KERNEL_BC);
   if (!ltp)
    goto free_mem_out;
   memset(ltp, 0, sizeof(struct termios));
@@ -1528,7 +1528,7 @@ static int init_dev(struct tty_driver *d

   if (!*o_tp_loc) {
    o_tp = (struct termios *)
-    kmalloc(sizeof(struct termios), GFP_KERNEL);
+    kmalloc(sizeof(struct termios), GFP_KERNEL_BC);
    if (!o_tp)
     goto free_mem_out;
    *o_tp = driver->other->init_termios;
@@ -1536,7 +1536,7 @@ static int init_dev(struct tty_driver *d

   if (!*o_ltp_loc) {
    o_ltp = (struct termios *)
-    kmalloc(sizeof(struct termios), GFP_KERNEL);
+    kmalloc(sizeof(struct termios), GFP_KERNEL_BC);
    if (!o_ltp)
     goto free_mem_out;
    memset(o_ltp, 0, sizeof(struct termios));
--- ./fs/file.c.bcslabs 2006-07-17 17:01:12.000000000 +0400
+++ ./fs/file.c 2006-08-01 15:18:03.000000000 +0400
@@ -44,9 +44,9 @@ struct file ** alloc_fd_array(int num)
  int size = num * sizeof(struct file *);

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

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

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

--- ./fs/locks.c.bcslabs 2006-07-10 12:39:16.000000000 +0400
+++ ./fs/locks.c 2006-08-01 12:46:47.000000000 +0400
@@ -2226,7 +2226,7 @@ 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.bcslabs 2006-07-10 12:39:16.000000000 +0400
+++ ./fs/namespace.c 2006-08-01 12:47:12.000000000 +0400
@@ -1825,7 +1825,8 @@ 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.bcslabs 2006-07-10 12:39:17.000000000 +0400
+++ ./fs/select.c 2006-08-01 15:17:01.000000000 +0400
@@ -103,7 +103,8 @@ static struct poll_table_entry *poll_get
  if (!table || POLL_TABLE_FULL(table)) {
   struct poll_table_page *new_table;

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

```
-  bits = kmalloc(6 * size, GFP_KERNEL);
+  bits = kmalloc(6 * size, GFP_KERNEL_BC);
   if (!bits)
    goto out_nofds;
  }
@@ -693,7 +694,7 @@ int do_sys_poll(struct pollfd __user *uf
   if (!stack_pp)
    stack_pp = pp = (struct poll_list *)stack_pps;
   else {
-   pp = kmalloc(size, GFP_KERNEL);
+   pp = kmalloc(size, GFP_KERNEL_BC);
    if (!pp)
     goto out_fds;
   }
--- ./include/asm-i386/thread_info.h.bcslabs 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-i386/thread_info.h 2006-08-01 15:19:50.000000000 +0400
@@ -99,13 +99,13 @@ static inline struct thread_info *curren
  ({       \
   struct thread_info *ret;   \
        \
-  ret = kmalloc(THREAD_SIZE, GFP_KERNEL);  \
+  ret = kmalloc(THREAD_SIZE, GFP_KERNEL_BC); \
   if (ret)     \
    memset(ret, 0, THREAD_SIZE);  \
   ret;      \
  })
 #else
-#define alloc_thread_info(tsk) kmalloc(THREAD_SIZE, GFP_KERNEL)
+#define alloc_thread_info(tsk) kmalloc(THREAD_SIZE, GFP_KERNEL_BC)
 #endif

 #define free_thread_info(info) kfree(info)
--- ./include/asm-ia64/pgalloc.h.bcslabs 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-ia64/pgalloc.h 2006-08-01 13:35:49.000000000 +0400
@@ -19,6 +19,8 @@
 #include <linux/page-flags.h>
 #include <linux/threads.h>

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

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

-static inline void *pgtable_quicklist_alloc(void)
```

```
+static inline void *pgtable_quicklist_alloc(int charge)
 {
  unsigned long *ret = NULL;

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

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

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

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

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

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

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

```
+ return pgtable_quicklist_alloc(1);
 }

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

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

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

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

 static inline void pte_free(struct page *pte)
--- ./include/asm-x86_64/pgalloc.h.bcslabs 2006-04-21 11:59:36.000000000 +0400
+++ ./include/asm-x86_64/pgalloc.h 2006-08-01 13:30:46.000000000 +0400
@@ -31,12 +31,14 @@ static inline void pmd_free(pmd_t *pmd)

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

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

 static inline void pud_free (pud_t *pud)
```

```
@@ -74,7 +76,8 @@ static inline void pgd_list_del(pgd_t *p
 static inline pgd_t *pgd_alloc(struct mm_struct *mm)
 {
  unsigned boundary;
- pgd_t *pgd = (pgd_t *)__get_free_page(GFP_KERNEL|__GFP_REPEAT);
+ pgd_t *pgd = (pgd_t *)__get_free_page(GFP_KERNEL|__GFP_REPEAT|
+    __GFP_BC | __GFP_BC_LIMIT);
  if (!pgd)
   return NULL;
  pgd_list_add(pgd);
@@ -105,7 +108,8 @@ static inline pte_t *pte_alloc_one_kerne

 static inline struct page *pte_alloc_one(struct mm_struct *mm, unsigned long address)
 {
- void *p = (void *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT);
+ void *p = (void *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT|
+    __GFP_BC | __GFP_BC_LIMIT);
  if (!p)
   return NULL;
  return virt_to_page(p);
--- ./include/asm-x86_64/thread_info.h.bcslabs 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-x86_64/thread_info.h 2006-08-01 15:20:30.000000000 +0400
@@ -78,14 +78,15 @@ static inline struct thread_info *stack_
     ({       \
  struct thread_info *ret;    \
         \
- ret = ((struct thread_info *) __get_free_pages(GFP_KERNEL,THREAD_ORDER)); \
+ ret = ((struct thread_info *) __get_free_pages(GFP_KERNEL_BC, \
+    THREAD_ORDER));   \
  if (ret)      \
   memset(ret, 0, THREAD_SIZE);   \
  ret;       \
     })
 #else
 #define alloc_thread_info(tsk) \
- ((struct thread_info *) __get_free_pages(GFP_KERNEL,THREAD_ORDER))
+ ((struct thread_info *) __get_free_pages(GFP_KERNEL_BC,THREAD_ORDER))
 #endif

 #define free_thread_info(ti) free_pages((unsigned long) (ti), THREAD_ORDER)
--- ./ipc/msgutil.c.bcslabs 2006-04-21 11:59:36.000000000 +0400
+++ ./ipc/msgutil.c 2006-08-01 15:22:58.000000000 +0400
@@ -36,7 +36,7 @@ struct msg_msg *load_msg(const void __us
  if (alen > DATALEN_MSG)
   alen = DATALEN_MSG;

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

```
  if (msg == NULL)
   return ERR_PTR(-ENOMEM);

@@ -57,7 +57,7 @@ struct msg_msg *load_msg(const void __us
   if (alen > DATALEN_SEG)
    alen = DATALEN_SEG;
   seg = (struct msg_msgseg *)kmalloc(sizeof(*seg) + alen,
-       GFP_KERNEL);
+       GFP_KERNEL_BC);
   if (seg == NULL) {
    err = -ENOMEM;
    goto out_err;
--- ./ipc/sem.c.bcslabs 2006-07-10 12:39:19.000000000 +0400
+++ ./ipc/sem.c 2006-08-01 15:22:33.000000000 +0400
@@ -954,7 +954,7 @@ 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);
@@ -1018,7 +1019,8 @@ static struct sem_undo *find_undo(int se
  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);
@@ -1076,7 +1078,7 @@ 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.bcslabs 2006-07-10 12:39:19.000000000 +0400
+++ ./ipc/util.c 2006-08-01 15:18:45.000000000 +0400
@@ -302,9 +302,9 @@ 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;
 }

@@ -387,14 +387,14 @@ 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.bcslabs 2006-07-31 18:40:20.000000000 +0400
+++ ./kernel/fork.c 2006-08-01 12:58:36.000000000 +0400
@@ -134,7 +134,7 @@ 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

  /*
@@ -1425,23 +1425,24 @@ 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.bcslabs 2006-04-21 11:59:36.000000000 +0400
+++ ./kernel/posix-timers.c 2006-08-01 12:58:57.000000000 +0400
@@ -242,7 +242,8 @@ static __init int init_posix_timers(void
  register_posix_clock(CLOCK_MONOTONIC, &clock_monotonic);

  posix_timers_cache = kmem_cache_create("posix_timers_cache",
-     sizeof (struct k_itimer), 0, 0, NULL, NULL);
+     sizeof (struct k_itimer), 0, SLAB_BC,
+     NULL, NULL);
  idr_init(&posix_timers_id);
  return 0;
 }
--- ./kernel/signal.c.bcslabs 2006-07-10 12:39:20.000000000 +0400
+++ ./kernel/signal.c 2006-08-01 12:59:14.000000000 +0400
@@ -2574,5 +2574,5 @@ 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.bcslabs 2006-07-10 12:39:20.000000000 +0400
+++ ./kernel/user.c 2006-08-01 12:59:38.000000000 +0400
@@ -197,7 +197,7 @@ static int __init uid_cache_init(void)
  int n;

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

```
+   0, SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);

  for(n = 0; n < UIDHASH_SZ; ++n)
   INIT_LIST_HEAD(uidhash_table + n);
--- ./mm/rmap.c.bcslabs 2006-07-10 12:39:20.000000000 +0400
+++ ./mm/rmap.c 2006-08-01 12:55:44.000000000 +0400
@@ -179,7 +179,8 @@ static void anon_vma_ctor(void *data, st
 void __init anon_vma_init(void)
 {
  anon_vma_cachep = kmem_cache_create("anon_vma", sizeof(struct anon_vma),
-   0, SLAB_DESTROY_BY_RCU|SLAB_PANIC, anon_vma_ctor, NULL);
+   0, SLAB_DESTROY_BY_RCU|SLAB_PANIC|SLAB_BC,
+   anon_vma_ctor, NULL);
 }

 /*
--- ./mm/shmem.c.bcslabs 2006-07-10 12:39:20.000000000 +0400
+++ ./mm/shmem.c 2006-08-01 13:26:13.000000000 +0400
@@ -367,7 +367,8 @@ static swp_entry_t *shmem_swp_alloc(stru
   }

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

Subject: Re: [PATCH 2/6] BC: beancounters core (API)
Posted by Andi Kleen on Wed, 23 Aug 2006 11:37:48 GMT

View Forum Message <> Reply to Message

On Wednesday 23 August 2006 13:03, Kirill Korotaev wrote:

> +#ifdef CONFIG_BEANCOUNTERS
> +extern struct hlist_head bc_hash[];
> +extern spinlock_t bc_hash_lock;

I wonder who pokes into that hash from other files? Looks a bit dangerous.

> +void __put_beancounter(struct beancounter *bc);
> +static inline void put_beancounter(struct beancounter *bc)
> +{
> + __put_beancounter(bc);
> +}

The wrapper seems pointless too.

The file could use a overview comment what the various counter
types actually are.

> + bc_print_id(bc, uid, sizeof(uid));
> + printk(KERN_WARNING "BC %s %s warning: %s "

Doesn't this need some rate limiting? Or can it be only triggered
by code bugs?

> + bc = &default_beancounter;
> + memset(bc, 0, sizeof(default_beancounter));

You don't trust the BSS to be zero? @)

-Andi

Subject: Re: [PATCH 1/6] BC: kconfig
Posted by Alexey Dobriyan on Wed, 23 Aug 2006 13:04:52 GMT
View Forum Message <> Reply to Message

On Wed, Aug 23, 2006 at 03:01:55PM +0400, Kirill Korotaev wrote:
> --- ./kernel/bc/Kconfig.bckm
> +++ ./kernel/bc/Kconfig
> @@ -0,0 +1,25 @@
> +#
> +# Resource beancounters (BC)
> +#
> +# Copyright (C) 2006 OpenVZ. SWsoft Inc
> +
> +menu "User resources"
> +
> +config BEANCOUNTERS
> + bool "Enable resource accounting/control"
> + default n
> + help
> + This patch provides accounting and allows to configure
> + limits for user's consumption of exhaustible system resources.

If BC will be merged there will be no patch, so text should be
rewritten slightly.

> + The most important resource controlled by this patch is
> unswappable + memory (either mlock'ed or used by internal kernel
> structures and + buffers). The main goal of this patch is to

> protect processes
> + from running short of important resources because of an
> accidental
> + misbehavior of processes or malicious activity aiming to
> ``kill'' + the system. It's worth to mention that resource limits
> configured + by setrlimit(2) do not give an acceptable level of
> protection + because they cover only small fraction of resources
> and work on a + per-process basis. Per-process accounting doesn't
> prevent malicious
> + users from spawning a lot of resource-consuming processes.

Subject: Re: [PATCH 2/6] BC: beancounters core (API)
Posted by dev on Wed, 23 Aug 2006 13:25:42 GMT
View Forum Message <> Reply to Message

Andi Kleen wrote:
> On Wednesday 23 August 2006 13:03, Kirill Korotaev wrote:
>
>
>>+#ifdef CONFIG_BEANCOUNTERS
>>+extern struct hlist_head bc_hash[];
>>+extern spinlock_t bc_hash_lock;
>
>
> I wonder who pokes into that hash from other files? Looks a bit dangerous.
it was kernel/ub/proc.c with proc interface :)
however, we removed it from this patchset version, but forgot extern's...

will remove

>>+void __put_beancounter(struct beancounter *bc);
>>+static inline void put_beancounter(struct beancounter *bc)
>>+{
>>+ __put_beancounter(bc);
>>+}
>
>
> The wrapper seems pointless too.
yep, almost the same reason :)

> The file could use a overview comment what the various counter
> types actually are.
you mean comment about what resource parameters we introduce?
ok, will add it with each resource patch.

>>+ bc_print_id(bc, uid, sizeof(uid));
>>+ printk(KERN_WARNING "BC %s %s warning: %s "

>
>
> Doesn't this need some rate limiting? Or can it be only triggered
> by code bugs?
only due to code bugs.

>>+ bc = &default_beancounter;
>>+ memset(bc, 0, sizeof(default_beancounter));
>
>
> You don't trust the BSS to be zero? @)
:))

Kirill

Subject: Re: [PATCH 2/6] BC: beancounters core (API)
Posted by Alexey Dobriyan on Wed, 23 Aug 2006 13:30:55 GMT
View Forum Message <> Reply to Message

On Wed, Aug 23, 2006 at 03:03:07PM +0400, Kirill Korotaev wrote:

> --- /dev/null
> +++ ./include/bc/beancounter.h

> +#define BC_RESOURCES 0

Do you want userspace to see it?

> +struct bc_resource_parm {
> + unsigned long barrier; /* A barrier over which resource allocations
> + * are failed gracefully. e.g. if the amount
> + * of consumed memory is over the barrier
> + * further sbrk() or mmap() calls fail, the
> + * existing processes are not killed.
> + */
> + unsigned long limit; /* hard resource limit */
> + unsigned long held; /* consumed resources */
> + unsigned long maxheld; /* maximum amount of consumed resources */
> + unsigned long minheld; /* minumum amount of consumed resources */

Stupid question: when minimum amount is useful?

> +/*
> + * Kernel internal part.
> + */

Redundant comment, YMMV.

> +#ifdef __KERNEL__
> +
> +#include <linux/config.h>

config.h is unneeded. You can drop it from everywhere.

> +struct beancounter
> +{

nit:
 "struct beancounter {"

> + atomic_t bc_refcount;
> + spinlock_t bc_lock;
> + uid_t   bc_id;
> + struct hlist_node hash;
> +
> + /* resources statistics and settings */
> + struct bc_resource_parm bc_parms[BC_RESOURCES];
> +};
> +
> +enum severity { BC_BARRIER, BC_LIMIT, BC_FORCE };

bc_severity?

> --- /dev/null 2006-07-18 14:52:43.075228448 +0400
> +++ ./kernel/bc/beancounter.c 2006-08-21 13:13:11.000000000 +0400

> +#define bc_hash_fun(x) ((((x) >> 8) ^ (x)) & (BC_HASH_SIZE - 1))
> +
> +struct hlist_head bc_hash[BC_HASH_SIZE];
> +spinlock_t bc_hash_lock;
> +
> +EXPORT_SYMBOL(bc_hash);
> +EXPORT_SYMBOL(bc_hash_lock);

tasklist_lock was unexported recently and this looks equally low-level.
I couldn't find place in patchbomb where you use it in modular fashion.
perhaps, i need more sleep.

> +void __put_beancounter(struct beancounter *bc)
> +{
> + unsigned long flags;
> +
> + /* equivalent to atomic_dec_and_lock_irqsave() */
> + local_irq_save(flags);
> + if (likely(!atomic_dec_and_lock(&bc->bc_refcount, &bc_hash_lock))) {

> + local_irq_restore(flags);
> + if (unlikely(atomic_read(&bc->bc_refcount) < 0))
> +   printk(KERN_ERR "BC: Bad refcount: bc=%p, "
> +   "luid=%d, ref=%d\n",
> +   bc, bc->bc_id,
> +   atomic_read(&bc->bc_refcount));

Should this BUG_ON() ?

> + return;
> + }
> +
> + BUG_ON(bc == &init_bc);
> + verify_held(bc);
> + hlist_del(&bc->hash);
> + spin_unlock_irqrestore(&bc_hash_lock, flags);
> + kmem_cache_free(bc_cachep, bc);
> +}
> +
> +EXPORT_SYMBOL(__put_beancounter);

> +int bc_charge_locked(struct beancounter *bc, int resource, unsigned long
> val,
> + enum severity strict)
> +{
> + /*
> + * bc_value <= BC_MAXVALUE, value <= BC_MAXVALUE, and only one
> addition
> + * at the moment is possible so an overflow is impossible.
> + */
> + bc->bc_parms[resource].held += val;
> +
> + switch (strict) {
> + case BC_BARRIER:

nit: one tab less

Subject: Re: [PATCH 4/6] BC: user interface (syscalls)
Posted by dev on Wed, 23 Aug 2006 13:40:51 GMT
View Forum Message <> Reply to Message

Sorry, wrong patch was attached.
This one fixes fixed but missed SPARC hunks.

Add the following system calls for BC management:

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

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

```
 arch/i386/kernel/syscall_table.S |    4 +
 arch/ia64/kernel/entry.S         |    4 +
 arch/sparc/kernel/entry.S        |    2
 arch/sparc/kernel/systbls.S      |    6 +
 arch/sparc64/kernel/entry.S      |    2
 arch/sparc64/kernel/systbls.S    |   10 ++-
 include/asm-i386/unistd.h        |    6 +
 include/asm-ia64/unistd.h        |    6 +
 include/asm-powerpc/systbl.h     |    4 +
 include/asm-powerpc/unistd.h     |    6 +
 include/asm-sparc/unistd.h       |    4 +
 include/asm-sparc64/unistd.h     |    4 +
 include/asm-x86_64/unistd.h      |   10 ++-
 kernel/sys_ni.c                  |    6 +
 kernel/bc/Makefile               |    1
 kernel/bc/sys.c                  |  120 ++++++++++++++++++++++++++++++++++++++++
 16 files changed, 186 insertions(+), 9 deletions(-)

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

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

 #define curptr      g6

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

 /* These are just handy. */
 #define _SV save %sp, -STACKFRAME_SZ, %sp
--- ./arch/sparc/kernel/systbls.S.ve3 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/sparc/kernel/systbls.S 2006-08-21 14:30:43.000000000 +0400
@@ -78,7 +78,8 @@ sys_call_table:
 /*285*/ .long sys_mkdirat, sys_mknodat, sys_fchownat, sys_futimesat, sys_fstatat64
 /*290*/ .long sys_unlinkat, sys_renameat, sys_linkat, sys_symlinkat, sys_readlinkat
 /*295*/ .long sys_fchmodat, sys_faccessat, sys_pselect6, sys_ppoll, sys_unshare
-/*300*/ .long sys_set_robust_list, sys_get_robust_list
+/*300*/ .long sys_set_robust_list, sys_get_robust_list, sys_get_bcid, sys_set_bcid,
sys_set_bclimit
+/*305*/ .long sys_get_bcstat

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

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

 #define curptr      g6

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

  .text
  .align  32
--- ./arch/sparc64/kernel/systbls.S.ve3 2006-07-10 12:39:11.000000000 +0400
+++ ./arch/sparc64/kernel/systbls.S 2006-08-21 14:32:26.000000000 +0400
@@ -79,7 +79,8 @@ sys_call_table32:
```

```
  .word sys_mkdirat, sys_mknodat, sys_fchownat, compat_sys_futimesat, compat_sys_fstatat64
 /*290*/ .word sys_unlinkat, sys_renameat, sys_linkat, sys_symlinkat, sys_readlinkat
  .word sys_fchmodat, sys_faccessat, compat_sys_pselect6, compat_sys_ppoll, sys_unshare
-/*300*/ .word compat_sys_set_robust_list, compat_sys_get_robust_list
+/*300*/ .word compat_sys_set_robust_list, compat_sys_get_robust_list, sys_nis_syscall,
sys_nis_syscall, sys_nis_syscall
+ .word sys_nis_syscall

 #endif /* CONFIG_COMPAT */

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

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

 #ifdef __KERNEL__

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

 /*
```

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

 #ifdef __KERNEL__


-#define NR_syscalls   279 /* length of syscall table */
+#define NR_syscalls   283 /* length of syscall table */

 #define __ARCH_WANT_SYS_RT_SIGACTION

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

 #ifdef __KERNEL__

-#define __NR_syscalls  301
+#define __NR_syscalls  305

 #define __NR__exit __NR_exit
 #define NR_syscalls __NR_syscalls
--- ./include/asm-sparc/unistd.h.ve3 2006-07-10 12:39:19.000000000 +0400
```

```
+++ ./include/asm-sparc/unistd.h 2006-08-21 14:33:20.000000000 +0400
@@ -318,6 +318,10 @@
 #define __NR_unshare  299
 #define __NR_set_robust_list 300
 #define __NR_get_robust_list 301
+#define __NR_get_bcid  302
+#define __NR_set_bcid  303
+#define __NR_set_bclimit 304
+#define __NR_get_bcstat  305

 #ifdef __KERNEL__
 /* WARNING: You MAY NOT add syscall numbers larger than 301, since
--- ./include/asm-sparc64/unistd.h.ve3 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-sparc64/unistd.h 2006-08-21 14:34:10.000000000 +0400
@@ -320,6 +320,10 @@
 #define __NR_unshare  299
 #define __NR_set_robust_list 300
 #define __NR_get_robust_list 301
+#define __NR_get_bcid  302
+#define __NR_set_bcid  303
+#define __NR_set_bclimit 304
+#define __NR_get_bcstat  305

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

 #ifdef __KERNEL__

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

 #ifndef __NO_STUBS
--- ./kernel/sys_ni.c.ve3 2006-07-10 12:39:20.000000000 +0400
```

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

 obj-$(CONFIG_BEANCOUNTERS) += beancounter.o
 obj-$(CONFIG_BEANCOUNTERS) += misc.o
+obj-$(CONFIG_BEANCOUNTERS) += sys.o
--- ./kernel/bc/sys.c.ve3 2006-08-21 13:49:49.000000000 +0400
+++ ./kernel/bc/sys.c 2006-08-21 14:43:04.000000000 +0400
@@ -0,0 +1,120 @@
+/*
+ *  kernel/bc/sys.c
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#include <linux/config.h>
+#include <linux/sched.h>
+#include <asm/uaccess.h>
+
+#include <bc/beancounter.h>
+#include <bc/task.h>
+
+asmlinkage long sys_get_bcid(void)
+{
+ struct beancounter *bc;
+
+ bc = get_exec_bc();
+ return bc->bc_id;
+}
+
+asmlinkage long sys_set_bcid(uid_t id)
+{
+ int error;
+ struct beancounter *bc;
+ struct task_beancounter *task_bc;
```

```
+
+ task_bc = &current->task_bc;
+
+ /* You may only set an bc as root */
+ error = -EPERM;
+ if (!capable(CAP_SETUID))
+  goto out;
+
+ /* Ok - set up a beancounter entry for this user */
+ error = -ENOMEM;
+ bc = beancounter_findcreate(id, BC_ALLOC);
+ if (bc == NULL)
+  goto out;
+
+ /* install bc */
+ put_beancounter(task_bc->exec_bc);
+ task_bc->exec_bc = bc;
+ put_beancounter(task_bc->fork_bc);
+ task_bc->fork_bc = get_beancounter(bc);
+ error = 0;
+out:
+ return error;
+}
+
+asmlinkage long sys_set_bclimit(uid_t id, unsigned long resource,
+  unsigned long *limits)
+{
+ int error;
+ unsigned long flags;
+ struct beancounter *bc;
+ unsigned long new_limits[2];
+
+ error = -EPERM;
+ if(!capable(CAP_SYS_RESOURCE))
+  goto out;
+
+ error = -EINVAL;
+ if (resource >= BC_RESOURCES)
+  goto out;
+
+ error = -EFAULT;
+ if (copy_from_user(&new_limits, limits, sizeof(new_limits)))
+  goto out;
+
+ error = -EINVAL;
+ if (new_limits[0] > BC_MAXVALUE || new_limits[1] > BC_MAXVALUE)
+  goto out;
+
```

```
+ error = -ENOENT;
+ bc = beancounter_findcreate(id, 0);
+ if (bc == NULL)
+  goto out;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ bc->bc_parms[resource].barrier = new_limits[0];
+ bc->bc_parms[resource].limit = new_limits[1];
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+
+ put_beancounter(bc);
+ error = 0;
+out:
+ return error;
+}
+
+int sys_get_bcstat(uid_t id, unsigned long resource,
+  struct bc_resource_parm *uparm)
+{
+ int error;
+ unsigned long flags;
+ struct beancounter *bc;
+ struct bc_resource_parm parm;
+
+ error = -EINVAL;
+ if (resource >= BC_RESOURCES)
+  goto out;
+
+ error = -ENOENT;
+ bc = beancounter_findcreate(id, 0);
+ if (bc == NULL)
+  goto out;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ parm = bc->bc_parms[resource];
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+ put_beancounter(bc);
+
+ error = 0;
+ if (copy_to_user(uparm, &parm, sizeof(parm)))
+  error = -EFAULT;
+
+out:
+ return error;
+}
```

Subject: Re: [PATCH 4/6] BC: user interface (syscalls)
Posted by Alexey Dobriyan on Wed, 23 Aug 2006 13:41:47 GMT
View Forum Message <> Reply to Message

On Wed, Aug 23, 2006 at 03:06:05PM +0400, Kirill Korotaev wrote:
> --- ./kernel/bc/sys.c.ve3
> +++ ./kernel/bc/sys.c

> +asmlinkage long sys_set_bclimit(uid_t id, unsigned long resource,
> + unsigned long *limits)

  unsigned long __user *limits

> +{
> + int error;
> + unsigned long flags;
> + struct beancounter *bc;
> + unsigned long new_limits[2];
> +
> + error = -EPERM;
> + if(!capable(CAP_SYS_RESOURCE))
> + goto out;
> +
> + error = -EINVAL;
> + if (resource >= BC_RESOURCES)
> + goto out;
> +
> + error = -EFAULT;
> + if (copy_from_user(&new_limits, limits, sizeof(new_limits)))
> + goto out;

> +int sys_get_bcstat(uid_t id, unsigned long resource,
> + struct bc_resource_parm *uparm)

    __user *uparm
> +{
> + int error;
> + unsigned long flags;
> + struct beancounter *bc;
> + struct bc_resource_parm parm;
> +
> + error = -EINVAL;
> + if (resource > BC_RESOURCES)
> + goto out;
> +
> + error = -ENOENT;
> + bc = beancounter_findcreate(id, 0);
> + if (bc == NULL)
> + goto out;

> +
> + spin_lock_irqsave(&bc->bc_lock, flags);
> + parm = bc->bc_parms[resource];
> + spin_unlock_irqrestore(&bc->bc_lock, flags);
> + put_beancounter(bc);
> +
> + error = 0;
> + if (copy_to_user(uparm, &parm, sizeof(parm)))
> + error = -EFAULT;

Subject: Re: [PATCH 2/6] BC: beancounters core (API)
Posted by dev on Wed, 23 Aug 2006 13:46:45 GMT
View Forum Message <> Reply to Message

Alexey Dobriyan wrote:
> On Wed, Aug 23, 2006 at 03:03:07PM +0400, Kirill Korotaev wrote:
>
>
>>--- /dev/null
>>+++ ./include/bc/beancounter.h
>
>
>>+#define BC_RESOURCES 0
>
>
> Do you want userspace to see it?
yep.

>>+struct bc_resource_parm {
>>+ unsigned long barrier; /* A barrier over which resource allocations
>>+ * are failed gracefully. e.g. if the amount
>>+ * of consumed memory is over the barrier
>>+ * further sbrk() or mmap() calls fail, the
>>+ * existing processes are not killed.
>>+ */
>>+ unsigned long limit; /* hard resource limit */
>>+ unsigned long held; /* consumed resources */
>>+ unsigned long maxheld; /* maximum amount of consumed resources */
>>+ unsigned long minheld; /* minumum amount of consumed resources */
>
>
> Stupid question: when minimum amount is useful?
to monitor usage statistics (range of used resources).
this value will be usefull when ubstat will be added.
this field probably would be more logical to add later,
but since it is part of user space interface it is left here
for not changing API later.

>>+/*
>>+ * Kernel internal part.
>>+ */
>
>
> Redundant comment, YMMV.
>
>
>>+#ifdef __KERNEL__
>>+
>>+#include <linux/config.h>
>
>
> config.h is unneeded. You can drop it from everywhere.
ok.

>>+struct beancounter
>>+{
>
>
> nit:
> "struct beancounter {"
>
>
>>+ atomic_t bc_refcount;
>>+ spinlock_t bc_lock;
>>+ uid_t   bc_id;
>>+ struct hlist_node hash;
>>+
>>+ /* resources statistics and settings */
>>+ struct bc_resource_parm bc_parms[BC_RESOURCES];
>>+};
>>+
>>+enum severity { BC_BARRIER, BC_LIMIT, BC_FORCE };
>
>
> bc_severity?
ok

>
>
>>--- /dev/null 2006-07-18 14:52:43.075228448 +0400
>>+++ ./kernel/bc/beancounter.c 2006-08-21 13:13:11.000000000 +0400
>
>
>>+#define bc_hash_fun(x) ((((x) >> 8) ^ (x)) & (BC_HASH_SIZE - 1))
>>+

>>+struct hlist_head bc_hash[BC_HASH_SIZE];
>>+spinlock_t bc_hash_lock;
>>+
>>+EXPORT_SYMBOL(bc_hash);
>>+EXPORT_SYMBOL(bc_hash_lock);
>
>
> tasklist_lock was unexported recently and this looks equally low-level.
> I couldn't find place in patchbomb where you use it in modular fashion.
> perhaps, i need more sleep.
I answered to Andi that it is left from prev patch set which used it in proc.c
will be removed
thanks for noting this!!!

>>+void __put_beancounter(struct beancounter *bc)
>>+{
>>+ unsigned long flags;
>>+
>>+ /* equivalent to atomic_dec_and_lock_irqsave() */
>>+ local_irq_save(flags);
>>+ if (likely(!atomic_dec_and_lock(&bc->bc_refcount, &bc_hash_lock))) {
>>+ local_irq_restore(flags);
>>+ if (unlikely(atomic_read(&bc->bc_refcount) < 0))
>>+   printk(KERN_ERR "BC: Bad refcount: bc=%p, "
>>+   "luid=%d, ref=%d\n",
>>+   bc, bc->bc_id,
>>+   atomic_read(&bc->bc_refcount));
>
>
> Should this BUG_ON() ?
BUG_ON doesn't print much information :)
ok, will replace

>>+ return;
>>+ }
>>+
>>+ BUG_ON(bc == &init_bc);
>>+ verify_held(bc);
>>+ hlist_del(&bc->hash);
>>+ spin_unlock_irqrestore(&bc_hash_lock, flags);
>>+ kmem_cache_free(bc_cachep, bc);
>>+}
>>+
>>+EXPORT_SYMBOL(__put_beancounter);
>
>
>>+int bc_charge_locked(struct beancounter *bc, int resource, unsigned long
>>val,

>>+ enum severity strict)
>>+{
>>+ /*
>>+ * bc_value <= BC_MAXVALUE, value <= BC_MAXVALUE, and only one
>>addition
>>+ * at the moment is possible so an overflow is impossible.
>>+ */
>>+ bc->bc_parms[resource].held += val;
>>+
>>+ switch (strict) {
>>+ case BC_BARRIER:
>
>
> nit: one tab less
>
> -
> 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: [PATCH 2/6] BC: beancounters core (API)
Posted by Andi Kleen on Wed, 23 Aug 2006 13:48:05 GMT
View Forum Message <> Reply to Message

On Wednesday 23 August 2006 15:27, Kirill Korotaev wrote:
> Andi Kleen wrote:
> > On Wednesday 23 August 2006 13:03, Kirill Korotaev wrote:
> >
> >
> > >>+#ifdef CONFIG_BEANCOUNTERS
> > >>+extern struct hlist_head bc_hash[];
> > >>+extern spinlock_t bc_hash_lock;
> >
> >
> > > I wonder who pokes into that hash from other files? Looks a bit dangerous.
> > it was kernel/ub/proc.c with proc interface :)
> > however, we removed it from this patchset version, but forgot extern's...
> >
> > will remove

Best remove the EXPORT_SYMBOLs too and make it static.

> > >>+void __put_beancounter(struct beancounter *bc);
> > >>+static inline void put_beancounter(struct beancounter *bc)
> > >>+{

> >>+ __put_beancounter(bc);
> >>+}
> >
> >
> > The wrapper seems pointless too.
> yep, almost the same reason :)
>
> > The file could use a overview comment what the various counter
> > types actually are.
> you mean comment about what resource parameters we introduce?

I meant about what a barrier counter etc. is and what makes
it different from other counters.

The individual resources can be probably described elswhere.

-Andi

Subject: Re: [PATCH 2/6] BC: beancounters core (API)
Posted by Alexey Dobriyan on Wed, 23 Aug 2006 13:53:11 GMT
View Forum Message <> Reply to Message

> >>+void __put_beancounter(struct beancounter *bc)
> >>+{
> >>+ unsigned long flags;
> >>+
> >>+ /* equivalent to atomic_dec_and_lock_irqsave() */
> >>+ local_irq_save(flags);
> >>+ if (likely(!atomic_dec_and_lock(&bc->bc_refcount, &bc_hash_lock))) {
> >>+ local_irq_restore(flags);
> >>+ if (unlikely(atomic_read(&bc->bc_refcount) < 0))
> >>+   printk(KERN_ERR "BC: Bad refcount: bc=%p, "
> >>+   "luid=%d, ref=%d\n",
> >>+   bc, bc->bc_id,
> >>+   atomic_read(&bc->bc_refcount));
> >
> >
> >Should this BUG_ON() ?
> BUG_ON doesn't print much information :)
> ok, will replace

but printk + BUG does ;-)

Subject: Re: [PATCH 2/6] BC: beancounters core (API)
Posted by Andrew Morton on Wed, 23 Aug 2006 16:42:02 GMT

On Wed, 23 Aug 2006 15:03:07 +0400
Kirill Korotaev <dev@sw.ru> wrote:

> Core functionality and interfaces of BC:
> find/create beancounter, initialization,
> charge/uncharge of resource, core objects' declarations.
>
> Basic structures:
> bc_resource_parm - resource description
> beancounter - set of resources, id, lock
>
>
> ..
>
> +enum severity { BC_BARRIER, BC_LIMIT, BC_FORCE };

That's a bit generic-sounding. Make it bc_severity?

> +static inline void bc_adjust_held_minmax(struct beancounter *bc,
> + int resource)
> +{
> + if (bc->bc_parms[resource].maxheld < bc->bc_parms[resource].held)
> + bc->bc_parms[resource].maxheld = bc->bc_parms[resource].held;
> + if (bc->bc_parms[resource].minheld > bc->bc_parms[resource].held)
> + bc->bc_parms[resource].minheld = bc->bc_parms[resource].held;
> +}

That might be a bit big to inline.

Suggest you check that the compiler successfully CSE's the
`bc->bc_parms[resource]' evaulation. If not, create a temporary.

> +#define beancounter_findcreate(id, f) (NULL)
> +#define get_beancounter(bc) (NULL)
> +#define put_beancounter(bc) do { } while (0)
> +#define bc_charge_locked(bc, r, v, s) (0)
> +#define bc_charge(bc, r, v) (0)

akpm:/home/akpm> cat t.c
void foo(void)
{
 (0);
}
akpm:/home/akpm> gcc -c -Wall t.c
t.c: In function 'foo':
t.c:4: warning: statement with no effect

> +#define bc_hash_fun(x) ((((x) >> 8) ^ (x)) & (BC_HASH_SIZE - 1))

Use hash_long()?

> +/*
> + * Per resource beancounting. Resources are tied to their luid.
> + * The resource structure itself is tagged both to the process and
> + * the charging resources (a socket doesn't want to have to search for
> + * things at irq time for example). Reference counters keep things in
> + * hand.
> + *
> + * The case where a user creates resource, kills all his processes and
> + * then starts new ones is correctly handled this way. The refcounters
> + * will mean the old entry is still around with resource tied to it.
> + */
> +
> +struct beancounter *beancounter_findcreate(uid_t uid, int mask)
> +{
> + struct beancounter *new_bc, *bc;
> + unsigned long flags;
> + struct hlist_head *slot;
> + struct hlist_node *pos;
> +
> + slot = &bc_hash[bc_hash_fun(uid)];
> + new_bc = NULL;
> +
> +retry:
> + spin_lock_irqsave(&bc_hash_lock, flags);
> + hlist_for_each_entry (bc, pos, slot, hash)
> + if (bc->bc_id == uid)
> +   break;
> +
> + if (pos != NULL) {
> + get_beancounter(bc);
> + spin_unlock_irqrestore(&bc_hash_lock, flags);
> +
> + if (new_bc != NULL)
> +   kmem_cache_free(bc_cachep, new_bc);
> + return bc;
> + }
> +
> + if (!(mask & BC_ALLOC))
> + goto out_unlock;
> +
> + if (new_bc != NULL)
> + goto out_install;
> +
> + spin_unlock_irqrestore(&bc_hash_lock, flags);

```
> +
> + new_bc = kmem_cache_alloc(bc_cachep,
> +   mask & BC_ALLOC_ATOMIC ? GFP_ATOMIC : GFP_KERNEL);
> + if (new_bc == NULL)
> +  goto out;
> +
> + memcpy(new_bc, &default_beancounter, sizeof(*new_bc));
> + init_beancounter_struct(new_bc, uid);
> + goto retry;
> +
> +out_install:
> + hlist_add_head(&new_bc->hash, slot);
> +out_unlock:
> + spin_unlock_irqrestore(&bc_hash_lock, flags);
> +out:
> + return new_bc;
> +}
```

Can remove the global bc_hash_lock and make the locking per-hash-bucket.

```
> +static inline void verify_held(struct beancounter *bc)
> +{
> + int i;
> +
> + for (i = 0; i < BC_RESOURCES; i++)
> +  if (bc->bc_parms[i].held != 0)
> +   bc_print_resource_warning(bc, i,
> +     "resource is held on put", 0, 0);
> +}
> +
> +void __put_beancounter(struct beancounter *bc)
> +{
> + unsigned long flags;
> +
> + /* equivalent to atomic_dec_and_lock_irqsave() */
> + local_irq_save(flags);
> + if (likely(!atomic_dec_and_lock(&bc->bc_refcount, &bc_hash_lock))) {
> +  local_irq_restore(flags);
> +  if (unlikely(atomic_read(&bc->bc_refcount) < 0))
> +   printk(KERN_ERR "BC: Bad refcount: bc=%p, "
> +     "luid=%d, ref=%d\n",
> +     bc, bc->bc_id,
> +     atomic_read(&bc->bc_refcount));
> +  return;
> + }
> +
> + BUG_ON(bc == &init_bc);
> + verify_held(bc);
```

> + hlist_del(&bc->hash);
> + spin_unlock_irqrestore(&bc_hash_lock, flags);
> + kmem_cache_free(bc_cachep, bc);
> +}

I wonder if it's safe and worthwhile to optimise away the local_irq_save():

```
 if (atomic_dec_and_test(&bc->bc_refcount)) {
  spin_lock_irqsave(&bc_hash_lock, flags);
  if (atomic_read(&bc->bc_refcount) == 0) {
   free it
```

Subject: Re: [PATCH 4/6] BC: user interface (syscalls)
Posted by Andrew Morton on Wed, 23 Aug 2006 16:50:31 GMT
View Forum Message <> Reply to Message

On Wed, 23 Aug 2006 17:43:16 +0400
Kirill Korotaev <dev@sw.ru> wrote:

> +asmlinkage long sys_set_bclimit(uid_t id, unsigned long resource,
> + unsigned long *limits)

I'm still a bit mystified about the use of uid_t here. It's not a uid, is
it?

Subject: Re: [PATCH] BC: resource beancounters (v2)
Posted by Andrew Morton on Wed, 23 Aug 2006 17:05:32 GMT
View Forum Message <> Reply to Message

On Wed, 23 Aug 2006 14:46:19 +0400
Kirill Korotaev <dev@sw.ru> wrote:

> The following patch set presents base of
> Resource Beancounters (BC).
> BC allows to account and control consumption
> of kernel resources used by group of processes.
>
> Draft UBC description on OpenVZ wiki can be found at
> http://wiki.openvz.org/UBC_parameters
>
> The full BC patch set allows to control:
> - kernel memory. All the kernel objects allocatable
> on user demand should be accounted and limited
> for DoS protection.
> E.g. page tables, task structs, vmas etc.

>
> - virtual memory pages. BCs allow to
> limit a container to some amount of memory and
> introduces 2-level OOM killer taking into account
> container's consumption.
> pages shared between containers are correctly
> charged as fractions (tunable).
>
> - network buffers. These includes TCP/IP rcv/snd
> buffers, dgram snd buffers, unix, netlinks and
> other buffers.
>
> - minor resources accounted/limited by number:
> tasks, files, flocks, ptys, siginfo, pinned dcache
> mem, sockets, iptentries (for containers with
> virtualized networking)
>
> As the first step we want to propose for discussion
> the most complicated parts of resource management:
> kernel memory and virtual memory.

The patches look reasonable to me - mergeable after updating them for
today's batch of review commentlets.

I have two high-level problems though.

a) I don't yet have a sense of whether this implementation
   is appropriate/sufficient for the various other
   applications which people are working on.

   If the general shape is OK and we think this
   implementation can be grown into one which everyone can
   use then fine.

And...

> The patch set to be sent provides core for BC and
> management of kernel memory only. Virtual memory
> management will be sent in a couple of days.

We need to go over this work before we can commit to the BC
core. Last time I looked at the VM accounting patch it
seemed rather unpleasing from a maintainability POV.

And, if I understand it correctly, the only response to a job
going over its VM limits is to kill it, rather than trimming
it. Which sounds like a big problem?

Subject: Re: [PATCH 4/6] BC: user interface (syscalls)
Posted by Alan Cox on Wed, 23 Aug 2006 17:08:28 GMT
View Forum Message <> Reply to Message

Ar Mer, 2006-08-23 am 09:50 -0700, ysgrifennodd Andrew Morton:
> On Wed, 23 Aug 2006 17:43:16 +0400
> Kirill Korotaev <dev@sw.ru> wrote:
>
> > +asmlinkage long sys_set_bclimit(uid_t id, unsigned long resource,
> > + unsigned long *limits)
>
> I'm still a bit mystified about the use of uid_t here. It's not a uid, is
> it?

Its a uid_t because of setluid() and twenty odd years of existing unix
practice.

Alan

Subject: Re: [PATCH 6/6] BC: kernel memory accounting (marks)
Posted by Dave Hansen on Wed, 23 Aug 2006 18:30:20 GMT
View Forum Message <> Reply to Message

I'm still a bit concerned about if we actually need the 'struct page'
pointer. I've gone through all of the users, and I'm not sure that I
see any that _require_ having a pointer in 'struct page'. I think it
will take some rework, especially with the pagetables, but it should be
quite doable.

vmalloc:
 Store in vm_struct
fd_set_bits:
poll_get:
mount hashtable:
 Don't need alignment. use the slab?
pagetables:
 either store in an extra field of 'struct page', or use the
 mm's. mm should always be available when alloc/freeing a
 pagetable page

Did I miss any?

-- Dave

Subject: Re: [PATCH] BC: resource beancounters (v2)

Posted by Cedric Le Goater on Wed, 23 Aug 2006 21:00:31 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
>
> [...]
>
> Patch set is applicable to 2.6.18-rc4-mm2

Applying patch 1_6_BC_kconfig.patch
patching file arch/i386/Kconfig
Hunk #1 succeeded at 1184 with fuzz 2 (offset 38 lines).
patching file arch/ia64/Kconfig
Hunk #1 succeeded at 505 with fuzz 2 (offset 24 lines).
patching file arch/i386/Kconfig
Hunk #1 succeeded at 1186 with fuzz 2 (offset 40 lines).
patching file arch/ia64/Kconfig
Hunk #1 succeeded at 507 with fuzz 2 (offset 26 lines).
patching file arch/powerpc/Kconfig
Hunk #1 succeeded at 1040 with fuzz 2 (offset 2 lines).
patching file arch/ppc/Kconfig
Hunk #1 succeeded at 1416 with fuzz 2 (offset 2 lines).
patching file arch/sparc/Kconfig
Hunk #1 FAILED at 296.
1 out of 1 hunk FAILED -- rejects in file arch/sparc/Kconfig
patching file arch/sparc64/Kconfig
Hunk #1 FAILED at 432.
1 out of 1 hunk FAILED -- rejects in file arch/sparc64/Kconfig
patching file arch/x86_64/Kconfig
Hunk #1 FAILED at 655.
1 out of 1 hunk FAILED -- rejects in file arch/x86_64/Kconfig
patching file kernel/bc/Kconfig
patching file arch/powerpc/Kconfig
Hunk #1 succeeded at 1042 with fuzz 2 (offset 4 lines).
patching file arch/ppc/Kconfig
Hunk #1 succeeded at 1418 with fuzz 2 (offset 4 lines).
patching file arch/sparc/Kconfig
Hunk #1 FAILED at 296.
1 out of 1 hunk FAILED -- rejects in file arch/sparc/Kconfig
patching file arch/sparc64/Kconfig
Hunk #1 FAILED at 432.
1 out of 1 hunk FAILED -- rejects in file arch/sparc64/Kconfig
patching file arch/x86_64/Kconfig
Hunk #1 FAILED at 655.
1 out of 1 hunk FAILED -- rejects in file arch/x86_64/Kconfig
patching file kernel/bc/Kconfig
Patch 1_6_BC_kconfig.patch does not apply (enforce with -f)

C.

Subject: Re: [PATCH 1/6] BC: kconfig
Posted by Dave Hansen on Wed, 23 Aug 2006 22:04:58 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-23 at 15:01 +0400, Kirill Korotaev wrote:
> --- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
> +++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
> @@ -432,3 +432,5 @@ source "security/Kconfig"
> source "crypto/Kconfig"
>
> source "lib/Kconfig"
> +
> +source "kernel/bc/Kconfig"
...
> --- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
> +++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
> @@ -432,3 +432,5 @@ source "security/Kconfig"
> source "crypto/Kconfig"
>
> source "lib/Kconfig"
> +
> +source "kernel/bc/Kconfig"

Is it just me, or do these patches look a little funky? Looks like it
is trying to patch the same thing into the same file, twice. Also, the
patches look to be -p0 instead of -p1.

I'm having a few problems applying them.

-- Dave

Subject: Re: [PATCH 2/6] BC: beancounters core (API)
Posted by Matt Helsley on Wed, 23 Aug 2006 22:05:55 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-23 at 17:49 +0400, Kirill Korotaev wrote:
> Alexey Dobriyan wrote:
> > On Wed, Aug 23, 2006 at 03:03:07PM +0400, Kirill Korotaev wrote:
> >
> >
> >>--- /dev/null
> >>+++ ./include/bc/beancounter.h
> >
> >
> >>+#define BC_RESOURCES 0
> >
> >

> > Do you want userspace to see it?
> yep.
>
> >>+struct bc_resource_parm {
> >>+ unsigned long barrier; /* A barrier over which resource allocations
> >>+ * are failed gracefully. e.g. if the amount
> >>+ * of consumed memory is over the barrier
> >>+ * further sbrk() or mmap() calls fail, the
> >>+ * existing processes are not killed.
> >>+ */
> >>+ unsigned long limit; /* hard resource limit */
> >>+ unsigned long held; /* consumed resources */
> >>+ unsigned long maxheld; /* maximum amount of consumed resources */
> >>+ unsigned long minheld; /* minumum amount of consumed resources */
> >
> >
> > Stupid question: when minimum amount is useful?
> to monitor usage statistics (range of used resources).
> this value will be usefull when ubstat will be added.
> this field probably would be more logical to add later,
> but since it is part of user space interface it is left here
> for not changing API later.

Then I think it belongs in a separate patch. Add it and the scattered bits and pieces that use it with the same patch. Then folks can clearly see what it's for, where it impacts the code, and how it works. Yes, factoring it out causes the API to evolve over the course of applying the patch series -- IMHO that evolution is useful information to convey to reviewers too.

Cheers,
-Matt Helsley

Subject: Re: [PATCH 1/6] BC: kconfig
Posted by Matt Helsley on Wed, 23 Aug 2006 22:13:42 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-23 at 15:04 -0700, Dave Hansen wrote:
> On Wed, 2006-08-23 at 15:01 +0400, Kirill Korotaev wrote:
> > --- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
> > +++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
> > @@ -432,3 +432,5 @@ source "security/Kconfig"
> > source "crypto/Kconfig"
> >
> > source "lib/Kconfig"
> > +
> > +source "kernel/bc/Kconfig"

> ...
> > --- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
> > +++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
> > @@ -432,3 +432,5 @@ source "security/Kconfig"
> > source "crypto/Kconfig"
> >
> > source "lib/Kconfig"
> > +
> > +source "kernel/bc/Kconfig"
>
> Is it just me, or do these patches look a little funky? Looks like it
> is trying to patch the same thing into the same file, twice. Also, the
> patches look to be -p0 instead of -p1.

They do appear to be -p0

 They aren't adding the same thing twice to the same file. This patch
makes different arches source the same Kconfig.

 I seem to recall Chandra suggested that instead of doing it this way it
would be more appropriate to add the source line to init/Kconfig because
it's more central and arch-independent. I tend to agree.

Cheers,
 -Matt Helsley

Subject: Re: [PATCH 1/6] BC: kconfig
Posted by Matt Helsley on Wed, 23 Aug 2006 22:27:59 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-23 at 15:32 -0700, Randy.Dunlap wrote:
> On Wed, 23 Aug 2006 15:13:42 -0700 Matt Helsley wrote:
>
> > On Wed, 2006-08-23 at 15:04 -0700, Dave Hansen wrote:
> > > On Wed, 2006-08-23 at 15:01 +0400, Kirill Korotaev wrote:
> > > > --- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
> > > > +++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
> > > > @@ -432,3 +432,5 @@ source "security/Kconfig"
> > > > source "crypto/Kconfig"
> > > >
> > > > source "lib/Kconfig"
> > > > +
> > > > +source "kernel/bc/Kconfig"
> > > ...
> > > > --- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
> > > > +++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
> > > > @@ -432,3 +432,5 @@ source "security/Kconfig"

> > > > source "crypto/Kconfig"
> > > >
> > > > source "lib/Kconfig"
> > > > +
> > > > +source "kernel/bc/Kconfig"
> > >
> > > Is it just me, or do these patches look a little funky? Looks like it
> > > is trying to patch the same thing into the same file, twice. Also, the
> > > patches look to be -p0 instead of -p1.
> >
> > They do appear to be -p0
> >
> > They aren't adding the same thing twice to the same file. This patch
> > makes different arches source the same Kconfig.
>
> Look again. There are 2 diffstat blocks and 2 of these at least:
>
> --- ./kernel/bc/Kconfig.bckm 2006-07-28 13:07:38.000000000 +0400
> +++ ./kernel/bc/Kconfig 2006-07-28 13:09:51.000000000 +0400
> @@ -0,0 +1,25 @@
>
>
> > I seem to recall Chandra suggested that instead of doing it this way it
> > would be more appropriate to add the source line to init/Kconfig because
> > it's more central and arch-independent. I tend to agree.
>
> ---
> ~Randy

My mistake, Dave is right.

Cheers,
 -Matt Helsley

Subject: Re: [PATCH 1/6] BC: kconfig
Posted by rdunlap on Wed, 23 Aug 2006 22:29:09 GMT
View Forum Message <> Reply to Message

On Wed, 23 Aug 2006 15:13:42 -0700 Matt Helsley wrote:

> On Wed, 2006-08-23 at 15:04 -0700, Dave Hansen wrote:
> > On Wed, 2006-08-23 at 15:01 +0400, Kirill Korotaev wrote:
> > > --- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
> > > +++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
> > > @@ -432,3 +432,5 @@ source "security/Kconfig"
> > > source "crypto/Kconfig"
> > >

> > > source "lib/Kconfig"
> > > +
> > > +source "kernel/bc/Kconfig"
> > ...
> > > --- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
> > > +++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
> > > @@ -432,3 +432,5 @@ source "security/Kconfig"
> > > source "crypto/Kconfig"
> > >
> > > source "lib/Kconfig"
> > > +
> > > +source "kernel/bc/Kconfig"
> >
> > Is it just me, or do these patches look a little funky? Looks like it
> > is trying to patch the same thing into the same file, twice. Also, the
> > patches look to be -p0 instead of -p1.
>
> They do appear to be -p0
>
> They aren't adding the same thing twice to the same file. This patch
> makes different arches source the same Kconfig.

Look again. There are 2 diffstat blocks and 2 of these at least:

--- ./kernel/bc/Kconfig.bckm 2006-07-28 13:07:38.000000000 +0400
+++ ./kernel/bc/Kconfig 2006-07-28 13:09:51.000000000 +0400
@@ -0,0 +1,25 @@

> I seem to recall Chandra suggested that instead of doing it this way it
> would be more appropriate to add the source line to init/Kconfig because
> it's more central and arch-independent. I tend to agree.

~Randy

Subject: Re: [PATCH 6/6] BC: kernel memory accounting (marks)
Posted by Dave Hansen on Wed, 23 Aug 2006 23:03:51 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-23 at 15:08 +0400, Kirill Korotaev wrote:
> 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 +++--

Do you think we need to cover a few more architectures before

considering merging this, or should we just fix them up as we need them?

I'm working on a patch to unify as many of the alloc_thread_info()
functions as I can. That should at least give you one place to modify
and track the thread_info allocations. I've only compiled for x86_64
and i386, but I'm working on more. A preliminary version is attached.

-- Dave

---

 clean-dave/include/asm-alpha/thread_info.h | 5 ---
 clean-dave/include/asm-frv/thread_info.h | 17 ----------
 clean-dave/include/asm-h8300/thread_info.h | 8 +----
 clean-dave/include/asm-i386/thread_info.h | 18 -----------
 clean-dave/include/asm-m32r/thread_info.h | 17 ----------
 clean-dave/include/asm-m68k/thread_info.h | 9 -----
 clean-dave/include/asm-powerpc/thread_info.h | 27 ------------------
 clean-dave/include/linux/thread_alloc.h | 42 +++++++++++++++++++++++++++
 clean-dave/kernel/fork.c | 1
 9 files changed, 47 insertions(+), 97 deletions(-)

diff -puN arch/arm/kernel/process.c~unify-alloc-thread-info arch/arm/kernel/process.c
diff -puN include/asm-alpha/thread_info.h~unify-alloc-thread-info include/asm-alpha/thread_info.h
--- clean/include/asm-alpha/thread_info.h~unify-alloc-thread-inf o 2006-08-23
15:38:37.000000000 -0700
+++ clean-dave/include/asm-alpha/thread_info.h 2006-08-23 15:40:23.000000000 -0700
@@ -50,10 +50,7 @@ register struct thread_info *__current_t
 #define current_thread_info() __current_thread_info

 /* Thread information allocation. */
-#define THREAD_SIZE (2*PAGE_SIZE)
-#define alloc_thread_info(tsk) \
- ((struct thread_info *) __get_free_pages(GFP_KERNEL,1))
-#define free_thread_info(ti) free_pages((unsigned long) (ti), 1)
+#define THREAD_SHIFT (PAGE_SHIFT+1)

 #endif /* __ASSEMBLY__ */

diff -puN include/asm-frv/thread_info.h~unify-alloc-thread-info include/asm-frv/thread_info.h
--- clean/include/asm-frv/thread_info.h~unify-alloc-thread-info 2006-08-23 15:40:26.000000000
-0700
+++ clean-dave/include/asm-frv/thread_info.h 2006-08-23 15:40:40.000000000 -0700
@@ -82,23 +82,6 @@ register struct thread_info *__current_t

 #define current_thread_info() ({ __current_thread_info; })

-/* thread information allocation */

```
-#ifdef CONFIG_DEBUG_STACK_USAGE
-#define alloc_thread_info(tsk)     \
- ({       \
-  struct thread_info *ret;   \
-        \
-  ret = kmalloc(THREAD_SIZE, GFP_KERNEL);  \
-  if (ret)     \
-   memset(ret, 0, THREAD_SIZE);  \
-  ret;      \
- })
-#else
-#define alloc_thread_info(tsk) kmalloc(THREAD_SIZE, GFP_KERNEL)
-#endif
-
-#define free_thread_info(info) kfree(info)
-
 #endif /* __ASSEMBLY__ */

 /*
diff -puN include/asm-h8300/thread_info.h~unify-alloc-thread-info
include/asm-h8300/thread_info.h
---  clean/include/asm-h8300/thread_info.h~unify-alloc-thread-inf o 2006-08-23
15:40:43.000000000 -0700
+++ clean-dave/include/asm-h8300/thread_info.h 2006-08-23 15:41:54.000000000 -0700
@@ -49,8 +49,8 @@ struct thread_info {
 /*
  * Size of kernel stack for each process. This must be a power of 2...
  */
-#define THREAD_SIZE  8192 /* 2 pages */
-
+#define THREAD_SHIFT 1
+#define THREAD_SIZE (1<<THREAD_SHIFT)

 /* how to get the thread information struct from C */
 static inline struct thread_info *current_thread_info(void)
@@ -65,10 +65,6 @@ static inline struct thread_info *curren
  return ti;
 }

-/* thread information allocation */
-#define alloc_thread_info(tsk) ((struct thread_info *) \
-    __get_free_pages(GFP_KERNEL, 1))
-#define free_thread_info(ti) free_pages((unsigned long) (ti), 1)
 #endif /* __ASSEMBLY__ */

 /*
diff -puN include/asm-i386/thread_info.h~unify-alloc-thread-info include/asm-i386/thread_info.h
---  clean/include/asm-i386/thread_info.h~unify-alloc-thread-info 2006-08-23 15:19:27.000000000
```

```
-0700
+++ clean-dave/include/asm-i386/thread_info.h 2006-08-23 15:33:30.000000000 -0700
@@ -92,24 +92,6 @@ static inline struct thread_info *curren
 {
  return (struct thread_info *)(current_stack_pointer & ~(THREAD_SIZE - 1));
 }
-
-/* thread information allocation */
-#ifdef CONFIG_DEBUG_STACK_USAGE
-#define alloc_thread_info(tsk)     \
- ({       \
-  struct thread_info *ret;   \
-        \
-  ret = kmalloc(THREAD_SIZE, GFP_KERNEL);  \
-  if (ret)     \
-   memset(ret, 0, THREAD_SIZE);  \
-  ret;      \
- })
-#else
-#define alloc_thread_info(tsk) kmalloc(THREAD_SIZE, GFP_KERNEL)
-#endif
-
-#define free_thread_info(info) kfree(info)
-
 #else /* !__ASSEMBLY__ */

 /* how to get the thread information struct from ASM */
diff -puN include/asm-ia64/thread_info.h~unify-alloc-thread-info include/asm-ia64/thread_info.h
diff -puN include/asm-m32r/thread_info.h~unify-alloc-thread-info include/asm-m32r/thread_info.h
---  clean/include/asm-m32r/thread_info.h~unify-alloc-thread-info 2006-08-23 15:44:38.000000000
-0700
+++ clean-dave/include/asm-m32r/thread_info.h 2006-08-23 15:44:51.000000000 -0700
@@ -94,23 +94,6 @@ static inline struct thread_info *curren
  return ti;
 }

-/* thread information allocation */
-#ifdef CONFIG_DEBUG_STACK_USAGE
-#define alloc_thread_info(tsk)     \
- ({       \
-  struct thread_info *ret;   \
-        \
-   ret = kmalloc(THREAD_SIZE, GFP_KERNEL);  \
-   if (ret)     \
-    memset(ret, 0, THREAD_SIZE);  \
-   ret;      \
-  })
-#else
```

```
-#define alloc_thread_info(tsk) kmalloc(THREAD_SIZE, GFP_KERNEL)
-#endif
-
-#define free_thread_info(info) kfree(info)
-
 #define TI_FLAG_FAULT_CODE_SHIFT 28

 static inline void set_thread_fault_code(unsigned int val)
diff -puN include/asm-m68k/thread_info.h~unify-alloc-thread-info include/asm-m68k/thread_info.h
---  clean/include/asm-m68k/thread_info.h~unify-alloc-thread-info 2006-08-23
15:44:52.000000000 -0700
+++ clean-dave/include/asm-m68k/thread_info.h 2006-08-23 15:45:32.000000000 -0700
@@ -24,14 +24,7 @@ struct thread_info {
  },     \
 }

-/* THREAD_SIZE should be 8k, so handle differently for 4k and 8k machines */
-#if PAGE_SHIFT == 13 /* 8k machines */
-#define alloc_thread_info(tsk)   ((struct thread_info *)__get_free_pages(GFP_KERNEL,0))
-#define free_thread_info(ti)  free_pages((unsigned long)(ti),0)
-#else /* otherwise assume 4k pages */
-#define alloc_thread_info(tsk)   ((struct thread_info *)__get_free_pages(GFP_KERNEL,1))
-#define free_thread_info(ti)  free_pages((unsigned long)(ti),1)
-#endif /* PAGE_SHIFT == 13 */
+#define THREAD_SHIFT 1

 #define init_thread_info (init_task.thread.info)
 #define init_stack  (init_thread_union.stack)
diff -puN include/asm-powerpc/thread_info.h~unify-alloc-thread-info
include/asm-powerpc/thread_info.h
---  clean/include/asm-powerpc/thread_info.h~unify-alloc-thread-i nfo 2006-08-07
12:21:11.000000000 -0700
+++ clean-dave/include/asm-powerpc/thread_info.h 2006-08-23 15:48:09.000000000 -0700
@@ -62,33 +62,6 @@ struct thread_info {
 #define init_thread_info (init_thread_union.thread_info)
 #define init_stack  (init_thread_union.stack)

-/* thread information allocation */
-
-#if THREAD_SHIFT >= PAGE_SHIFT
-
-#define THREAD_ORDER (THREAD_SHIFT - PAGE_SHIFT)
-
-#ifdef CONFIG_DEBUG_STACK_USAGE
-#define alloc_thread_info(tsk) \
- ((struct thread_info *)__get_free_pages(GFP_KERNEL | \
-  __GFP_ZERO, THREAD_ORDER))
-#else
```

```
-#define alloc_thread_info(tsk) \
- ((struct thread_info *)__get_free_pages(GFP_KERNEL, THREAD_ORDER))
-#endif
-#define free_thread_info(ti) free_pages((unsigned long)ti, THREAD_ORDER)
-
-#else /* THREAD_SHIFT < PAGE_SHIFT */
-
-#ifdef CONFIG_DEBUG_STACK_USAGE
-#define alloc_thread_info(tsk) kzalloc(THREAD_SIZE, GFP_KERNEL)
-#else
-#define alloc_thread_info(tsk) kmalloc(THREAD_SIZE, GFP_KERNEL)
-#endif
-#define free_thread_info(ti) kfree(ti)
-
-#endif /* THREAD_SHIFT < PAGE_SHIFT */
-
 /* how to get the thread information struct from C */
 static inline struct thread_info *current_thread_info(void)
 {
diff -puN /dev/null include/linux/thread_alloc.h
--- /dev/null 2005-03-30 22:36:15.000000000 -0800
+++ clean-dave/include/linux/thread_alloc.h 2006-08-23 16:00:41.000000000 -0700
@@ -0,0 +1,42 @@
+#ifndef _LINUX_THREAD_ALLOC
+#define _LINUX_THREAD_ALLOC
+
+#ifndef THREAD_SHIFT
+#define THREAD_SHIFT PAGE_SHIFT
+#endif
+#ifndef THREAD_ORDER
+#define THREAD_ORDER    (THREAD_SHIFT - PAGE_SHIFT)
+#endif
+
+struct thread_info;
+struct task;
+
+#if THREAD_SHIFT >= PAGE_SHIFT
+static inline struct thread_info *alloc_thread_info(struct task_struct *tsk)
+{
+ gfp_t flags = GFP_KERNEL;
+#ifdef CONFIG_DEBUG_STACK_USAGE
+ flags |= __GFP_ZERO;
+#endif
+ return (struct thread_info *)__get_free_pages(flags, THREAD_ORDER);
+}
+static inline void free_thread_info(struct thread_info *ti)
+{
+ free_pages((unsigned long)ti, THREAD_ORDER);
```

```
+}
+#else /* THREAD_SHIFT < PAGE_SHIFT */
+static inline struct thread_info *alloc_thread_info(struct task_struct *tsk)
+{
+#ifdef CONFIG_DEBUG_STACK_USAGE
+ return kzalloc(THREAD_SIZE, GFP_KERNEL);
+#else
+ return kmalloc(THREAD_SIZE, GFP_KERNEL);
+#endif
+}
+static inline void free_thread_info(struct thread_info *ti)
+{
+ kfree(ti);
+}
+#endif /* THREAD_SHIFT < PAGE_SHIFT */
+
+#endif /* _LINUX_THREAD_ALLOC */
diff -puN include/linux/thread_info.h~unify-alloc-thread-info include/linux/thread_info.h
diff -puN kernel/fork.c~unify-alloc-thread-info kernel/fork.c
--- clean/kernel/fork.c~unify-alloc-thread-info 2006-08-23 15:19:28.000000000 -0700
+++ clean-dave/kernel/fork.c 2006-08-23 15:47:35.000000000 -0700
@@ -45,6 +45,7 @@
 #include <linux/cn_proc.h>
 #include <linux/delayacct.h>
 #include <linux/taskstats_kern.h>
+#include <linux/thread_alloc.h>

 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
_
```

Subject: Re: [PATCH] BC: resource beancounters (v2)
Posted by Chandra Seetharaman on Thu, 24 Aug 2006 00:17:39 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-23 at 10:05 -0700, Andrew Morton wrote:
> On Wed, 23 Aug 2006 14:46:19 +0400
> Kirill Korotaev <dev@sw.ru> wrote:
>
> > The following patch set presents base of
> > Resource Beancounters (BC).
> > BC allows to account and control consumption
> > of kernel resources used by group of processes.
> >
> > Draft UBC description on OpenVZ wiki can be found at
> > http://wiki.openvz.org/UBC_parameters
> >

> > The full BC patch set allows to control:
> > - kernel memory. All the kernel objects allocatable
> > on user demand should be accounted and limited
> > for DoS protection.
> > E.g. page tables, task structs, vmas etc.
> >
> > - virtual memory pages. BCs allow to
> > limit a container to some amount of memory and
> > introduces 2-level OOM killer taking into account
> > container's consumption.
> > pages shared between containers are correctly
> > charged as fractions (tunable).
> >
> > - network buffers. These includes TCP/IP rcv/snd
> > buffers, dgram snd buffers, unix, netlinks and
> > other buffers.
> >
> > - minor resources accounted/limited by number:
> > tasks, files, flocks, ptys, siginfo, pinned dcache
> > mem, sockets, iptentries (for containers with
> > virtualized networking)
> >
> > As the first step we want to propose for discussion
> > the most complicated parts of resource management:
> > kernel memory and virtual memory.
>
> The patches look reasonable to me - mergeable after updating them for
> today's batch of review commentlets.

If you are considering this infrastructure for generic resource
management, I have few concerns:
 - There is no CPU controller under this framework
 - There is no I/O controller under this framework
 - Minimum of 3 parameters need to be used to manage memory.
   (in other words, usage is not simple. In order to provide a minimum
    guarantee of a resource, one needs to define a new parameter)

>
> I have two high-level problems though.
>
> a) I don't yet have a sense of whether this implementation
> is appropriate/sufficient for the various other
> applications which people are working on.
>
> If the general shape is OK and we think this
> implementation can be grown into one which everyone can
> use then fine.

Here are some of other infrastructure related issues I have raised.
http://marc.theaimsgroup.com/?l=ckrm-tech&m=115593001810 616&w=2

>
> And...
>
> > The patch set to be sent provides core for BC and
> > management of kernel memory only. Virtual memory
> > management will be sent in a couple of days.
>
> We need to go over this work before we can commit to the BC
> core. Last time I looked at the VM accounting patch it
> seemed rather unpleasing from a maintainability POV.
>
> And, if I understand it correctly, the only response to a job
> going over its VM limits is to kill it, rather than trimming
> it. Which sounds like a big problem?

Yes, it does.

IMHO (as mentioned in a different email), a group with a resource constraint should behave no different than a kernel with a specified amount of memory. i.e it should do reclamation before it starts failing allocation requests. It could even do it preemptively.
>
--

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

Subject: Re: [PATCH 1/6] BC: kconfig
Posted by Chandra Seetharaman on Thu, 24 Aug 2006 00:23:48 GMT
View Forum Message <> Reply to Message

Is there a reason why these can be moved to a arch-neutral place ?

PS: Please keep ckrm-tech on Cc: please.

On Wed, 2006-08-23 at 15:01 +0400, Kirill Korotaev wrote:
> 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>
>

> ---
>
> arch/i386/Kconfig | 2 ++
> arch/ia64/Kconfig | 2 ++
> arch/powerpc/Kconfig | 2 ++
> arch/ppc/Kconfig | 2 ++
> arch/sparc/Kconfig | 2 ++
> arch/sparc64/Kconfig | 2 ++
> arch/x86_64/Kconfig | 2 ++
> kernel/bc/Kconfig | 25 ++++++++++++++++++++++++
> 8 files changed, 39 insertions(+)
>
> --- ./arch/i386/Kconfig.bckm 2006-07-10 12:39:10.000000000 +0400
> +++ ./arch/i386/Kconfig 2006-07-28 14:10:41.000000000 +0400
> @@ -1146,6 +1146,8 @@ source "crypto/Kconfig"
>
> source "lib/Kconfig"
>
> +source "kernel/bc/Kconfig"
> +
> #
> # Use the generic interrupt handling code in kernel/irq/:
> #
> --- ./arch/ia64/Kconfig.bckm 2006-07-10 12:39:10.000000000 +0400
> +++ ./arch/ia64/Kconfig 2006-07-28 14:10:56.000000000 +0400
> @@ -481,6 +481,8 @@ source "fs/Kconfig"
>
> source "lib/Kconfig"
>
> +source "kernel/bc/Kconfig"
> +
> #
> # Use the generic interrupt handling code in kernel/irq/:
> #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>
>
> ---
>
> arch/i386/Kconfig | 2 ++
> arch/ia64/Kconfig | 2 ++
> arch/powerpc/Kconfig | 2 ++
> arch/ppc/Kconfig | 2 ++
> arch/sparc/Kconfig | 2 ++
> arch/sparc64/Kconfig | 2 ++
> arch/x86_64/Kconfig | 2 ++

> kernel/bc/Kconfig | 25 ++++++++++++++++++++++++
> 8 files changed, 39 insertions(+)
>
> --- ./arch/i386/Kconfig.bckm 2006-07-10 12:39:10.000000000 +0400
> +++ ./arch/i386/Kconfig 2006-07-28 14:10:41.000000000 +0400
> @@ -1146,6 +1146,8 @@ source "crypto/Kconfig"
>
> source "lib/Kconfig"
>
> +source "kernel/bc/Kconfig"
> +
> #
> # Use the generic interrupt handling code in kernel/irq/:
> #
> --- ./arch/ia64/Kconfig.bckm 2006-07-10 12:39:10.000000000 +0400
> +++ ./arch/ia64/Kconfig 2006-07-28 14:10:56.000000000 +0400
> @@ -481,6 +481,8 @@ source "fs/Kconfig"
>
> source "lib/Kconfig"
>
> +source "kernel/bc/Kconfig"
> +
> #
> # Use the generic interrupt handling code in kernel/irq/:
> #
> --- ./arch/powerpc/Kconfig.arkcfg 2006-08-07 14:07:12.000000000 +0400
> +++ ./arch/powerpc/Kconfig 2006-08-10 17:55:58.000000000 +0400
> @@ -1038,6 +1038,8 @@ source "arch/powerpc/platforms/iseries/K
>
> source "lib/Kconfig"
>
> +source "kernel/bc/Kconfig"
> +
> menu "Instrumentation Support"
>   depends on EXPERIMENTAL
>
> --- ./arch/ppc/Kconfig.arkcfg 2006-07-10 12:39:10.000000000 +0400
> +++ ./arch/ppc/Kconfig 2006-08-10 17:56:13.000000000 +0400
> @@ -1414,6 +1414,8 @@ endmenu
>
> source "lib/Kconfig"
>
> +source "kernel/bc/Kconfig"
> +
> source "arch/powerpc/oprofile/Kconfig"
>
> source "arch/ppc/Kconfig.debug"
> --- ./arch/sparc/Kconfig.arkcfg 2006-04-21 11:59:32.000000000 +0400

> +++ ./arch/sparc/Kconfig 2006-08-10 17:56:24.000000000 +0400
> @@ -296,3 +296,5 @@ source "security/Kconfig"
> source "crypto/Kconfig"
>
> source "lib/Kconfig"
> +
> +source "kernel/bc/Kconfig"
> --- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
> +++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
> @@ -432,3 +432,5 @@ source "security/Kconfig"
> source "crypto/Kconfig"
>
> source "lib/Kconfig"
> +
> +source "kernel/bc/Kconfig"
> --- ./arch/x86_64/Kconfig.bckm 2006-07-10 12:39:11.000000000 +0400
> +++ ./arch/x86_64/Kconfig 2006-07-28 14:10:49.000000000 +0400
> @@ -655,3 +655,5 @@ source "security/Kconfig"
> source "crypto/Kconfig"
>
> source "lib/Kconfig"
> +
> +source "kernel/bc/Kconfig"
> --- ./kernel/bc/Kconfig.bckm 2006-07-28 13:07:38.000000000 +0400
> +++ ./kernel/bc/Kconfig 2006-07-28 13:09:51.000000000 +0400
> @@ -0,0 +1,25 @@
> +#
> +# Resource beancounters (BC)
> +#
> +# Copyright (C) 2006 OpenVZ. SWsoft Inc
> +
> +menu "User resources"
> +
> +config BEANCOUNTERS
> + bool "Enable resource accounting/control"
> + default n
> + help
> + This patch provides accounting and allows to configure
> + limits for user's consumption of exhaustible system resources.
> + The most important resource controlled by this patch is unswappable
> + memory (either mlock'ed or used by internal kernel structures and
> + buffers). The main goal of this patch is to protect processes
> + from running short of important resources because of an accidental
> + misbehavior of processes or malicious activity aiming to ``kill''
> + the system. It's worth to mention that resource limits configured
> + by setrlimit(2) do not give an acceptable level of protection
> + because they cover only small fraction of resources and work on a
> + per-process basis. Per-process accounting doesn't prevent malicious

> +   users from spawning a lot of resource-consuming processes.
> +
> +endmenu
>
> --- ./arch/powerpc/Kconfig.arkcfg 2006-08-07 14:07:12.000000000 +0400
> +++ ./arch/powerpc/Kconfig 2006-08-10 17:55:58.000000000 +0400
> @@ -1038,6 +1038,8 @@ source "arch/powerpc/platforms/iseries/K
>
> source "lib/Kconfig"
>
> +source "kernel/bc/Kconfig"
> +
> menu "Instrumentation Support"
>   depends on EXPERIMENTAL
>
> --- ./arch/ppc/Kconfig.arkcfg 2006-07-10 12:39:10.000000000 +0400
> +++ ./arch/ppc/Kconfig 2006-08-10 17:56:13.000000000 +0400
> @@ -1414,6 +1414,8 @@ endmenu
>
> source "lib/Kconfig"
>
> +source "kernel/bc/Kconfig"
> +
> source "arch/powerpc/oprofile/Kconfig"
>
> source "arch/ppc/Kconfig.debug"
> --- ./arch/sparc/Kconfig.arkcfg 2006-04-21 11:59:32.000000000 +0400
> +++ ./arch/sparc/Kconfig 2006-08-10 17:56:24.000000000 +0400
> @@ -296,3 +296,5 @@ source "security/Kconfig"
> source "crypto/Kconfig"
>
> source "lib/Kconfig"
> +
> +source "kernel/bc/Kconfig"
> --- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
> +++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
> @@ -432,3 +432,5 @@ source "security/Kconfig"
> source "crypto/Kconfig"
>
> source "lib/Kconfig"
> +
> +source "kernel/bc/Kconfig"
> --- ./arch/x86_64/Kconfig.bckm 2006-07-10 12:39:11.000000000 +0400
> +++ ./arch/x86_64/Kconfig 2006-07-28 14:10:49.000000000 +0400
> @@ -655,3 +655,5 @@ source "security/Kconfig"
> source "crypto/Kconfig"
>
> source "lib/Kconfig"

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

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

Subject: Re: [PATCH 4/6] BC: user interface (syscalls)
Posted by Chandra Seetharaman on Thu, 24 Aug 2006 00:30:56 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-23 at 15:06 +0400, Kirill Korotaev wrote:
<snip>

> +asmlinkage long sys_set_bclimit(uid_t id, unsigned long resource,
> + unsigned long *limits)
> +{

```
> + int error;
> + unsigned long flags;
> + struct beancounter *bc;
> + unsigned long new_limits[2];
> +
> + error = -EPERM;
> + if(!capable(CAP_SYS_RESOURCE))
> +  goto out;
> +
> + error = -EINVAL;
> + if (resource >= BC_RESOURCES)
> +  goto out;
> +
> + error = -EFAULT;
> + if (copy_from_user(&new_limits, limits, sizeof(new_limits)))
> +  goto out;
> +
> + error = -EINVAL;
> + if (new_limits[0] > BC_MAXVALUE || new_limits[1] > BC_MAXVALUE)
> +  goto out;
> +
> + error = -ENOENT;
> + bc = beancounter_findcreate(id, 0);
> + if (bc == NULL)
> +  goto out;
> +
> + spin_lock_irqsave(&bc->bc_lock, flags);
> + bc->bc_parms[resource].barrier = new_limits[0];
> + bc->bc_parms[resource].limit = new_limits[1];

 No check for barrier <= limit
> + spin_unlock_irqrestore(&bc->bc_lock, flags);
> +
> + put_beancounter(bc);
> + error = 0;
> +out:
> + return error;
> +}
<snip>
--

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

Subject: Re: [PATCH 5/6] BC: kernel memory accounting (core)
Posted by Chandra Seetharaman on Thu, 24 Aug 2006 00:36:28 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-23 at 15:06 +0400, Kirill Korotaev wrote:

<snip>

> --- ./include/bc/beancounter.h.bckmem 2006-07-28 18:43:52.000000000 +0400
> +++ ./include/bc/beancounter.h 2006-08-03 16:03:01.000000000 +0400
> @@ -14,7 +14,9 @@
> * Resource list.
> */
>
> -#define BC_RESOURCES 0
> +#define BC_KMEMSIZE 0
> +
> +#define BC_RESOURCES 1

As suggested before, a clean interface to define these would be better
>
> struct resource_parm {
> /*
> --- ./include/bc/kmem.h.bckmem 2006-07-28 18:43:52.000000000 +0400
> +++ ./include/bc/kmem.h 2006-07-31 17:37:05.000000000 +0400
> @@ -0,0 +1,33 @@
> +/*
> + * include/bc/kmem.h
> + *
> + * Copyright (C) 2006 OpenVZ. SWsoft Inc
> + *
> + */
> +
> +#ifndef __BC_KMEM_H_
> +#define __BC_KMEM_H_
> +
> +#include <linux/config.h>
> +
> +/*
> + * BC_KMEMSIZE accounting
> + */
> +
> +struct mm_struct;
> +struct page;
> +struct beancounter;
> +
> +#ifdef CONFIG_BEANCOUNTERS
> +int bc_page_charge(struct page *page, int order, gfp_t flags);
> +void bc_page_uncharge(struct page *page, int order);

> +
> +int bc_slab_charge(kmem_cache_t *cachep, void *obj, gfp_t flags);
> +void bc_slab_uncharge(kmem_cache_t *cachep, void *obj);
> +#else
> +#define bc_page_charge(pg, o, mask) (0)
> +#define bc_page_uncharge(pg, o) do { } while (0)
> +#define bc_slab_charge(cachep, o, f) (0)
> +#define bc_slab_uncharge(cachep, o) do { } while (0)
> +#endif
> +#endif /* __BC_SLAB_H_ */
> --- ./kernel/bc/Makefile.bcsys 2006-07-28 14:08:37.000000000 +0400
> +++ ./kernel/bc/Makefile 2006-08-01 11:08:39.000000000 +0400
> @@ -7,3 +7,4 @@
> obj-$(CONFIG_BEANCOUNTERS) += beancounter.o
> obj-$(CONFIG_BEANCOUNTERS) += misc.o
> obj-$(CONFIG_BEANCOUNTERS) += sys.o
> +obj-$(CONFIG_BEANCOUNTERS) += kmem.o
> --- ./kernel/bc/beancounter.c.bckmem 2006-07-28 18:43:52.000000000 +0400
> +++ ./kernel/bc/beancounter.c 2006-08-03 16:14:17.000000000 +0400
> @@ -19,6 +19,7 @@ static void init_beancounter_struct(stru
> struct beancounter init_bc;
>
> const char *bc_rnames[] = {
> + "kmemsize", /* 0 */
> };

As suggested before, a clean interface would be a better idea.
>
> #define bc_hash_fun(x) ((((x) >> 8) ^ (x)) & (BC_HASH_SIZE - 1))
> @@ -348,6 +378,8 @@ static void init_beancounter_syslimits(s
> {
>   int k;
>
> + bc->bc_parms[BC_KMEMSIZE].limit = 32 * 1024 * 1024;
> +

This could be either be a configurable value by/for the controller or
can be set by the controller through a callout.

>   for (k = 0; k < BC_RESOURCES; k++)
>   bc->bc_parms[k].barrier = bc->bc_parms[k].limit;
> }

<snip>

--

 ---------- ----------

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

Subject: Re: [PATCH 4/6] BC: user interface (syscalls)
Posted by Andrew Morton on Thu, 24 Aug 2006 04:35:12 GMT
View Forum Message <> Reply to Message

On Wed, 23 Aug 2006 18:29:42 +0100
Alan Cox <alan@lxorguk.ukuu.org.uk> wrote:

> Ar Mer, 2006-08-23 am 09:50 -0700, ysgrifennodd Andrew Morton:
> > On Wed, 23 Aug 2006 17:43:16 +0400
> > Kirill Korotaev <dev@sw.ru> wrote:
> >
> > > +asmlinkage long sys_set_bclimit(uid_t id, unsigned long resource,
> > > + unsigned long *limits)
> >
> > I'm still a bit mystified about the use of uid_t here. It's not a uid, is
> > it?
>
> Its a uid_t because of setluid() and twenty odd years of existing unix
> practice.
>

I don't understand. This number is an identifier for an accounting
container, which was somehow dreamed up by userspace.

AFAICT it is wholly unrelated to user ID's.

(How does userspace avoid collisions, btw?)

Subject: Re: [PATCH] BC: resource beancounters (v2)
Posted by Jan Engelhardt on Thu, 24 Aug 2006 05:52:36 GMT
View Forum Message <> Reply to Message

>
> The following patch set presents base of
> Resource Beancounters (BC).

Why are they called beans?

Jan Engelhardt
--

Subject: Re: [PATCH 6/6] BC: kernel memory accounting (marks)
Posted by Geert Uytterhoeven on Thu, 24 Aug 2006 09:30:09 GMT
View Forum Message <> Reply to Message

On Wed, 23 Aug 2006, Dave Hansen wrote:
> I'm working on a patch to unify as many of the alloc_thread_info()
> functions as I can. That should at least give you one place to modify
> and track the thread_info allocations. I've only compiled for x86_64
> and i386, but I'm working on more. A preliminary version is attached.

> --- clean/include/asm-m68k/thread_info.h~unify-alloc-thread-info 2006-08-23
15:44:52.000000000 -0700
> +++ clean-dave/include/asm-m68k/thread_info.h 2006-08-23 15:45:32.000000000 -0700
> @@ -24,14 +24,7 @@ struct thread_info {
> }, \
> }
>
> -/* THREAD_SIZE should be 8k, so handle differently for 4k and 8k machines */
> -#if PAGE_SHIFT == 13 /* 8k machines */
> -#define alloc_thread_info(tsk) ((struct thread_info *)__get_free_pages(GFP_KERNEL,0))
> -#define free_thread_info(ti) free_pages((unsigned long)(ti),0)
> -#else /* otherwise assume 4k pages */
> -#define alloc_thread_info(tsk) ((struct thread_info *)__get_free_pages(GFP_KERNEL,1))
> -#define free_thread_info(ti) free_pages((unsigned long)(ti),1)
> -#endif /* PAGE_SHIFT == 13 */
> +#define THREAD_SHIFT 1
^
Shouldn't this be 13?

> --- /dev/null 2005-03-30 22:36:15.000000000 -0800
> +++ clean-dave/include/linux/thread_alloc.h 2006-08-23 16:00:41.000000000 -0700
> @@ -0,0 +1,42 @@
> +#ifndef _LINUX_THREAD_ALLOC
> +#define _LINUX_THREAD_ALLOC
> +
> +#ifndef THREAD_SHIFT
> +#define THREAD_SHIFT PAGE_SHIFT
> +#endif
> +#ifndef THREAD_ORDER
> +#define THREAD_ORDER (THREAD_SHIFT - PAGE_SHIFT)
> +#endif
> +
> +struct thread_info;
> +struct task;
> +
> +#if THREAD_SHIFT >= PAGE_SHIFT
> +static inline struct thread_info *alloc_thread_info(struct task_struct *tsk)
> +{
> + gfp_t flags = GFP_KERNEL;

```
> +#ifdef CONFIG_DEBUG_STACK_USAGE
> + flags |= __GFP_ZERO;
> +#endif
> + return (struct thread_info *)__get_free_pages(flags, THREAD_ORDER);
> +}
> +static inline void free_thread_info(struct thread_info *ti)
> +{
> + free_pages((unsigned long)ti, THREAD_ORDER);
> +}
> +#else /* THREAD_SHIFT < PAGE_SHIFT */
> +static inline struct thread_info *alloc_thread_info(struct task_struct *tsk)
> +{
> +#ifdef CONFIG_DEBUG_STACK_USAGE
> + return kzalloc(THREAD_SIZE, GFP_KERNEL);
> +#else
> + return kmalloc(THREAD_SIZE, GFP_KERNEL);
> +#endif
> +}
> +static inline void free_thread_info(struct thread_info *ti)
> +{
> + kfree(ti);
> +}
> +#endif /* THREAD_SHIFT < PAGE_SHIFT */
> +
> +#endif /* _LINUX_THREAD_ALLOC */
```

Gr{oetje,eeting}s,

Geert

--
Geert Uytterhoeven -- There's lots of Linux beyond ia32 -- geert@linux-m68k.org

In personal conversations with technical people, I call myself a hacker. But
when I'm talking to journalists I just say "programmer" or something like that.
-- Linus Torvalds

Subject: Re: [PATCH] BC: resource beancounters (v2)
Posted by Alan Cox on Thu, 24 Aug 2006 10:38:07 GMT
View Forum Message <> Reply to Message

Ar Iau, 2006-08-24 am 07:52 +0200, ysgrifennodd Jan Engelhardt:
> >
> > The following patch set presents base of
> > Resource Beancounters (BC).
>
> Why are they called beans?

"Beancounter" is a slang English language name for an accountant.

Alan

Subject: Re: [PATCH 4/6] BC: user interface (syscalls)
Posted by Alan Cox on Thu, 24 Aug 2006 10:42:55 GMT
View Forum Message <> Reply to Message

Ar Mer, 2006-08-23 am 21:35 -0700, ysgrifennodd Andrew Morton:
> > Its a uid_t because of setluid() and twenty odd years of existing unix
> > practice.
> >
>
> I don't understand. This number is an identifier for an accounting
> container, which was somehow dreamed up by userspace.

Which happens to be a uid_t. It could easily be anyother_t of itself and
you can create a container_id_t or whatever. It is just a number.

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. So we can certainly do something like

typedef uid_t container_id_t;

Alan

Subject: Re: [PATCH 2/6] BC: beancounters core (API)
Posted by Oleg Nesterov on Thu, 24 Aug 2006 11:09:01 GMT
View Forum Message <> Reply to Message

On 08/23, Andrew Morton wrote:
>
> On Wed, 23 Aug 2006 15:03:07 +0400
> Kirill Korotaev <dev@sw.ru> wrote:
>
> > +void __put_beancounter(struct beancounter *bc)

```
> > +{
> > + unsigned long flags;
> > +
> > + /* equivalent to atomic_dec_and_lock_irqsave() */
> > + local_irq_save(flags);
> > + if (likely(!atomic_dec_and_lock(&bc->bc_refcount, &bc_hash_lock))) {
> > +  local_irq_restore(flags);
> > +  if (unlikely(atomic_read(&bc->bc_refcount) < 0))
> > +   printk(KERN_ERR "BC: Bad refcount: bc=%p, "
> > +     "luid=%d, ref=%d\n",
> > +     bc, bc->bc_id,
> > +     atomic_read(&bc->bc_refcount));
> > +  return;
> > + }
> > +
> > + BUG_ON(bc == &init_bc);
> > + verify_held(bc);
> > + hlist_del(&bc->hash);
> > + spin_unlock_irqrestore(&bc_hash_lock, flags);
> > + kmem_cache_free(bc_cachep, bc);
> > +}
>
> I wonder if it's safe and worthwhile to optimise away the local_irq_save():
```

Suppose ->bc_refcount == 1

```
>  if (atomic_dec_and_test(&bc->bc_refcount)) {
```

Yes, preempted or blocks on spin_lock() below.

another cpu locks bc_hash_lock, does get_beancounter() (beancounter_findcreate),
then does put_beancounter(), and frees it.

```
>   spin_lock_irqsave(&bc_hash_lock, flags);
>   if (atomic_read(&bc->bc_refcount) == 0) {
```

Yes,

```
>    free it
>
```

Already freed.

Oleg.

Subject: Re: [ckrm-tech] [PATCH 1/6] BC: kconfig

> Is there a reason why these can be moved to a arch-neutral place ?
I think a good place for BC would be kernel/Kconfig.bc
but still this should be added into archs.
ok?

> PS: Please keep ckrm-tech on Cc: please.
Sorry, it is very hard to track emails coming from authors and 3 mailing lists.
Better tell me the interested people emails.

Thanks,
Kirill

>
> On Wed, 2006-08-23 at 15:01 +0400, Kirill Korotaev wrote:
>
>>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>
>>
>>---
>>
>> arch/i386/Kconfig | 2 ++
>> arch/ia64/Kconfig | 2 ++
>> arch/powerpc/Kconfig | 2 ++
>> arch/ppc/Kconfig | 2 ++
>> arch/sparc/Kconfig | 2 ++
>> arch/sparc64/Kconfig | 2 ++
>> arch/x86_64/Kconfig | 2 ++
>> kernel/bc/Kconfig | 25 ++++++++++++++++++++++++
>> 8 files changed, 39 insertions(+)
>>
>>--- ./arch/i386/Kconfig.bckm 2006-07-10 12:39:10.000000000 +0400
>>+++ ./arch/i386/Kconfig 2006-07-28 14:10:41.000000000 +0400
>>@@ -1146,6 +1146,8 @@ source "crypto/Kconfig"
>>
>> source "lib/Kconfig"
>>
>>+source "kernel/bc/Kconfig"
>>+
>> #
>> # Use the generic interrupt handling code in kernel/irq/:
>> #
>>--- ./arch/ia64/Kconfig.bckm 2006-07-10 12:39:10.000000000 +0400
>>+++ ./arch/ia64/Kconfig 2006-07-28 14:10:56.000000000 +0400

>>@@ -481,6 +481,8 @@ source "fs/Kconfig"
>>
>> source "lib/Kconfig"
>>
>>+source "kernel/bc/Kconfig"
>>+
>> #
>> # Use the generic interrupt handling code in kernel/irq/:
>> #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>
>>
>>---
>>
>> arch/i386/Kconfig | 2 ++
>> arch/ia64/Kconfig | 2 ++
>> arch/powerpc/Kconfig | 2 ++
>> arch/ppc/Kconfig | 2 ++
>> arch/sparc/Kconfig | 2 ++
>> arch/sparc64/Kconfig | 2 ++
>> arch/x86_64/Kconfig | 2 ++
>> kernel/bc/Kconfig | 25 ++++++++++++++++++++++++
>> 8 files changed, 39 insertions(+)
>>
>>--- ./arch/i386/Kconfig.bckm 2006-07-10 12:39:10.000000000 +0400
>>+++ ./arch/i386/Kconfig 2006-07-28 14:10:41.000000000 +0400
>>@@ -1146,6 +1146,8 @@ source "crypto/Kconfig"
>>
>> source "lib/Kconfig"
>>
>>+source "kernel/bc/Kconfig"
>>+
>> #
>> # Use the generic interrupt handling code in kernel/irq/:
>> #
>>--- ./arch/ia64/Kconfig.bckm 2006-07-10 12:39:10.000000000 +0400
>>+++ ./arch/ia64/Kconfig 2006-07-28 14:10:56.000000000 +0400
>>@@ -481,6 +481,8 @@ source "fs/Kconfig"
>>
>> source "lib/Kconfig"
>>
>>+source "kernel/bc/Kconfig"
>>+
>> #
>> # Use the generic interrupt handling code in kernel/irq/:
>> #

>>--- ./arch/powerpc/Kconfig.arkcfg 2006-08-07 14:07:12.000000000 +0400
>>+++ ./arch/powerpc/Kconfig 2006-08-10 17:55:58.000000000 +0400
>>@@ -1038,6 +1038,8 @@ source "arch/powerpc/platforms/iseries/K
>>
>> source "lib/Kconfig"
>>
>>+source "kernel/bc/Kconfig"
>>+
>> menu "Instrumentation Support"
>>   depends on EXPERIMENTAL
>>
>>--- ./arch/ppc/Kconfig.arkcfg 2006-07-10 12:39:10.000000000 +0400
>>+++ ./arch/ppc/Kconfig 2006-08-10 17:56:13.000000000 +0400
>>@@ -1414,6 +1414,8 @@ endmenu
>>
>> source "lib/Kconfig"
>>
>>+source "kernel/bc/Kconfig"
>>+
>> source "arch/powerpc/oprofile/Kconfig"
>>
>> source "arch/ppc/Kconfig.debug"
>>--- ./arch/sparc/Kconfig.arkcfg 2006-04-21 11:59:32.000000000 +0400
>>+++ ./arch/sparc/Kconfig 2006-08-10 17:56:24.000000000 +0400
>>@@ -296,3 +296,5 @@ source "security/Kconfig"
>> source "crypto/Kconfig"
>>
>> source "lib/Kconfig"
>>+
>>+source "kernel/bc/Kconfig"
>>--- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
>>+++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
>>@@ -432,3 +432,5 @@ source "security/Kconfig"
>> source "crypto/Kconfig"
>>
>> source "lib/Kconfig"
>>+
>>+source "kernel/bc/Kconfig"
>>--- ./arch/x86_64/Kconfig.bckm 2006-07-10 12:39:11.000000000 +0400
>>+++ ./arch/x86_64/Kconfig 2006-07-28 14:10:49.000000000 +0400
>>@@ -655,3 +655,5 @@ source "security/Kconfig"
>> source "crypto/Kconfig"
>>
>> source "lib/Kconfig"
>>+
>>+source "kernel/bc/Kconfig"
>>--- ./kernel/bc/Kconfig.bckm 2006-07-28 13:07:38.000000000 +0400
>>+++ ./kernel/bc/Kconfig 2006-07-28 13:09:51.000000000 +0400

```
>>@@ -0,0 +1,25 @@
>>+#
>>+# Resource beancounters (BC)
>>+#
>>+# Copyright (C) 2006 OpenVZ. SWsoft Inc
>>+
>>+menu "User resources"
>>+
>>+config BEANCOUNTERS
>>+ bool "Enable resource accounting/control"
>>+ default n
>>+ help
>>+         This patch provides accounting and allows to configure
>>+         limits for user's consumption of exhaustible system resources.
>>+         The most important resource controlled by this patch is unswappable
>>+         memory (either mlock'ed or used by internal kernel structures and
>>+         buffers). The main goal of this patch is to protect processes
>>+         from running short of important resources because of an accidental
>>+         misbehavior of processes or malicious activity aiming to ``kill''
>>+         the system. It's worth to mention that resource limits configured
>>+         by setrlimit(2) do not give an acceptable level of protection
>>+         because they cover only small fraction of resources and work on a
>>+         per-process basis.  Per-process accounting doesn't prevent malicious
>>+         users from spawning a lot of resource-consuming processes.
>>+
>>+endmenu
>>
>>--- ./arch/powerpc/Kconfig.arkcfg 2006-08-07 14:07:12.000000000 +0400
>>+++ ./arch/powerpc/Kconfig 2006-08-10 17:55:58.000000000 +0400
>>@@ -1038,6 +1038,8 @@ source "arch/powerpc/platforms/iseries/K
>>
>> source "lib/Kconfig"
>>
>>+source "kernel/bc/Kconfig"
>>+
>> menu "Instrumentation Support"
>>         depends on EXPERIMENTAL
>>
>>--- ./arch/ppc/Kconfig.arkcfg 2006-07-10 12:39:10.000000000 +0400
>>+++ ./arch/ppc/Kconfig 2006-08-10 17:56:13.000000000 +0400
>>@@ -1414,6 +1414,8 @@ endmenu
>>
>> source "lib/Kconfig"
>>
>>+source "kernel/bc/Kconfig"
>>+
>> source "arch/powerpc/oprofile/Kconfig"
>>
```

```
>> source "arch/ppc/Kconfig.debug"
>>--- ./arch/sparc/Kconfig.arkcfg 2006-04-21 11:59:32.000000000 +0400
>>+++ ./arch/sparc/Kconfig 2006-08-10 17:56:24.000000000 +0400
>>@@ -296,3 +296,5 @@ source "security/Kconfig"
>> source "crypto/Kconfig"
>>
>> source "lib/Kconfig"
>>+
>>+source "kernel/bc/Kconfig"
>>--- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
>>+++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
>>@@ -432,3 +432,5 @@ source "security/Kconfig"
>> source "crypto/Kconfig"
>>
>> source "lib/Kconfig"
>>+
>>+source "kernel/bc/Kconfig"
>>--- ./arch/x86_64/Kconfig.bckm 2006-07-10 12:39:11.000000000 +0400
>>+++ ./arch/x86_64/Kconfig 2006-07-28 14:10:49.000000000 +0400
>>@@ -655,3 +655,5 @@ source "security/Kconfig"
>> source "crypto/Kconfig"
>>
>> source "lib/Kconfig"
>>+
>>+source "kernel/bc/Kconfig"
>>--- ./kernel/bc/Kconfig.bckm 2006-07-28 13:07:38.000000000 +0400
>>+++ ./kernel/bc/Kconfig 2006-07-28 13:09:51.000000000 +0400
>>@@ -0,0 +1,25 @@
>>+#
>>+# Resource beancounters (BC)
>>+#
>>+# Copyright (C) 2006 OpenVZ. SWsoft Inc
>>+
>>+menu "User resources"
>>+
>>+config BEANCOUNTERS
>>+ bool "Enable resource accounting/control"
>>+ default n
>>+ help
>>+ This patch provides accounting and allows to configure
>>+ limits for user's consumption of exhaustible system resources.
>>+ The most important resource controlled by this patch is unswappable
>>+ memory (either mlock'ed or used by internal kernel structures and
>>+ buffers). The main goal of this patch is to protect processes
>>+ from running short of important resources because of an accidental
>>+ misbehavior of processes or malicious activity aiming to ``kill''
>>+ the system. It's worth to mention that resource limits configured
>>+ by setrlimit(2) do not give an acceptable level of protection
```

>>+ because they cover only small fraction of resources and work on a
>>+ per-process basis. Per-process accounting doesn't prevent malicious
>>+ users from spawning a lot of resource-consuming processes.
>>+
>>+endmenu

Subject: Re: [PATCH 2/6] BC: beancounters core (API)
Posted by dev on Thu, 24 Aug 2006 12:03:38 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:
[... snip ...]

>>+#define bc_charge_locked(bc, r, v, s) (0)
>>> +#define bc_charge(bc, r, v) (0)
>
>akpm:/home/akpm> cat t.c
>void foo(void)
>{
> (0);
>}
>akpm:/home/akpm> gcc -c -Wall t.c
>t.c: In function 'foo':
>t.c:4: warning: statement with no effect

these functions return value should always be checked (!).
i.e. it is never called like:
ub_charge(bc, r, v);

>>+struct beancounter *beancounter_findcreate(uid_t uid, int mask)
>>+{
>>+ struct beancounter *new_bc, *bc;
>>+ unsigned long flags;
>>+ struct hlist_head *slot;
>>+ struct hlist_node *pos;
>>+
>>+ slot = &bc_hash[bc_hash_fun(uid)];
>>+ new_bc = NULL;
>>+
>>+retry:
>>+ spin_lock_irqsave(&bc_hash_lock, flags);
>>+ hlist_for_each_entry (bc, pos, slot, hash)
>>+ if (bc->bc_id == uid)
>>+ break;
>>+
>>+ if (pos != NULL) {
>>+ get_beancounter(bc);

```
>>+  spin_unlock_irqrestore(&bc_hash_lock, flags);
>>+
>>+  if (new_bc != NULL)
>>+   kmem_cache_free(bc_cachep, new_bc);
>>+  return bc;
>>+ }
>>+
>>+ if (!(mask & BC_ALLOC))
>>+  goto out_unlock;
>>+
>>+ if (new_bc != NULL)
>>+  goto out_install;
>>+
>>+ spin_unlock_irqrestore(&bc_hash_lock, flags);
>>+
>>+ new_bc = kmem_cache_alloc(bc_cachep,
>>+   mask & BC_ALLOC_ATOMIC ? GFP_ATOMIC : GFP_KERNEL);
>>+ if (new_bc == NULL)
>>+  goto out;
>>+
>>+ memcpy(new_bc, &default_beancounter, sizeof(*new_bc));
>>+ init_beancounter_struct(new_bc, uid);
>>+ goto retry;
>>+
>>+out_install:
>>+ hlist_add_head(&new_bc->hash, slot);
>>+out_unlock:
>>+ spin_unlock_irqrestore(&bc_hash_lock, flags);
>>+out:
>>+ return new_bc;
>>+}
>
>
> Can remove the global bc_hash_lock and make the locking per-hash-bucket.
it is not performance critical path IMHO.
this lock is taken on container create/change/destroy/user interfaces only.

>>+static inline void verify_held(struct beancounter *bc)
>>+{
>>+ int i;
>>+
>>+ for (i = 0; i < BC_RESOURCES; i++)
>>+  if (bc->bc_parms[i].held != 0)
>>+   bc_print_resource_warning(bc, i,
>>+     "resource is held on put", 0, 0);
>>+}
>>+
>>+void __put_beancounter(struct beancounter *bc)
```

>>+{
>>+ unsigned long flags;
>>+
>>+ /* equivalent to atomic_dec_and_lock_irqsave() */
>>+ local_irq_save(flags);
>>+ if (likely(!atomic_dec_and_lock(&bc->bc_refcount, &bc_hash_lock))) {
>>+ local_irq_restore(flags);
>>+ if (unlikely(atomic_read(&bc->bc_refcount) < 0))
>>+   printk(KERN_ERR "BC: Bad refcount: bc=%p, "
>>+   "luid=%d, ref=%d\n",
>>+   bc, bc->bc_id,
>>+   atomic_read(&bc->bc_refcount));
>>+ return;
>>+ }
>>+
>>+ BUG_ON(bc == &init_bc);
>>+ verify_held(bc);
>>+ hlist_del(&bc->hash);
>>+ spin_unlock_irqrestore(&bc_hash_lock, flags);
>>+ kmem_cache_free(bc_cachep, bc);
>>+}
>
> I wonder if it's safe and worthwhile to optimise away the local_irq_save():
>
> if (atomic_dec_and_test(&bc->bc_refcount)) {
>   spin_lock_irqsave(&bc_hash_lock, flags);
>   if (atomic_read(&bc->bc_refcount) == 0) {
>   free it
put_beancounter can happen from IRQ context.
so we need something like atomic_dec_and_lock_irqsave()
Oleg Nesterov proposed more details.

Thanks,
Kirill

Subject: Re: [PATCH 4/6] BC: user interface (syscalls)
Posted by Alexey Dobriyan on Thu, 24 Aug 2006 13:08:22 GMT
View Forum Message <> Reply to Message

On Thu, Aug 24, 2006 at 12:04:16PM +0100, Alan Cox wrote:
> Ar Mer, 2006-08-23 am 21:35 -0700, ysgrifennodd Andrew Morton:
> > > Its a uid_t because of setluid() and twenty odd years of existing unix
> > > practice.
> > >
> >
> > I don't understand. This number is an identifier for an accounting
> > container, which was somehow dreamed up by userspace.

>
> Which happens to be a uid_t. It could easily be anyother_t of itself and
> you can create a container_id_t or whatever. It is just a number.
>
> 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. So we can certainly do something like
>
> typedef uid_t container_id_t;

What about cid_t? Google mentions cid_t was used in HP-UX specific IPC (only if _INCLUDE_HPUX_SOURCE is defined).

Subject: Re: [PATCH 2/6] BC: beancounters core (API)
Posted by Andrew Morton on Thu, 24 Aug 2006 15:00:30 GMT
View Forum Message <> Reply to Message

On Thu, 24 Aug 2006 16:06:11 +0400
Kirill Korotaev <dev@sw.ru> wrote:

> >>+#define bc_charge_locked(bc, r, v, s)   (0)
> >>> +#define bc_charge(bc, r, v)   (0)
> >
> >akpm:/home/akpm> cat t.c
> >void foo(void)
> >{
> > (0);
> >}
> >akpm:/home/akpm> gcc -c -Wall t.c
> >t.c: In function 'foo':
> >t.c:4: warning: statement with no effect
>
> these functions return value should always be checked (!).

We have __must_check for that.

> i.e. it is never called like:
>   ub_charge(bc, r, v);

Also...

if (bc_charge(tpyo, undefined_variable, syntax_error))

will happily compile if !CONFIG_BEANCOUNTER.

Turning these stubs into static inline __must_check functions fixes all this.

Subject: Re: [PATCH 6/6] BC: kernel memory accounting (marks)
Posted by Dave Hansen on Thu, 24 Aug 2006 15:52:22 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-24 at 11:30 +0200, Geert Uytterhoeven wrote:
> > +#define THREAD_SHIFT 1
> ^
> Shouldn't this be 13?

Yep. Thanks!

I need to go over the whole things again and proofread.

-- Dave

Subject: Re: [PATCH 5/6] BC: kernel memory accounting (core)
Posted by Oleg Nesterov on Thu, 24 Aug 2006 16:59:39 GMT
View Forum Message <> Reply to Message

On 08/23, Kirill Korotaev wrote:
>
> +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();
> + if (bc == NULL)
> + return 0;

Is it possible to .exec_bc == NULL ?

If yes, why do we need init_bc? We can do 'set_exec_bc(NULL)' in __do_IRQ()
instead.

Oleg.

Subject: Re: [PATCH 2/6] BC: beancounters core (API)
Posted by Oleg Nesterov on Thu, 24 Aug 2006 17:09:45 GMT
View Forum Message <> Reply to Message

On 08/23, Kirill Korotaev wrote:
>
> +struct beancounter *beancounter_findcreate(uid_t uid, int mask)
> +{
> + struct beancounter *new_bc, *bc;
> + unsigned long flags;
> + struct hlist_head *slot;
> + struct hlist_node *pos;
> +
> + slot = &bc_hash[bc_hash_fun(uid)];
> + new_bc = NULL;
> +
> +retry:
> + spin_lock_irqsave(&bc_hash_lock, flags);
> + hlist_for_each_entry (bc, pos, slot, hash)
> + if (bc->bc_id == uid)
> + break;
> +
> + if (pos != NULL) {
> + get_beancounter(bc);
> + spin_unlock_irqrestore(&bc_hash_lock, flags);
> +
> + if (new_bc != NULL)
> + kmem_cache_free(bc_cachep, new_bc);
> + return bc;
> + }
> +
> + if (!(mask & BC_ALLOC))
> + goto out_unlock;

Very minor nit: it is not clear why we are doing this check under
bc_hash_lock. I'd suggest to do

 if (!(mask & BC_ALLOC))
  goto out;

after unlock(bc_hash_lock) and kill out_unlock label.

> + if (new_bc != NULL)
> + goto out_install;
> +
> + spin_unlock_irqrestore(&bc_hash_lock, flags);
> +
> + new_bc = kmem_cache_alloc(bc_cachep,
> + mask & BC_ALLOC_ATOMIC ? GFP_ATOMIC : GFP_KERNEL);

> + if (new_bc == NULL)
> + goto out;
> +
> + memcpy(new_bc, &default_beancounter, sizeof(*new_bc));

May be it is just me, but I need a couple of seconds to parse this 'memcpy'.
How about

 *new_bc = default_beancounter;

?

Oleg.

Subject: Re: [ckrm-tech] [PATCH 1/6] BC: kconfig
Posted by Matt Helsley on Thu, 24 Aug 2006 22:23:31 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-24 at 15:47 +0400, Kirill Korotaev wrote:
> > Is there a reason why these can be moved to a arch-neutral place ?
> I think a good place for BC would be kernel/Kconfig.bc

I think kernel/bc/Kconfig is fine.

> but still this should be added into archs.
> ok?

Sourcing the bc Kconfig from arch Kconfigs would seem to suggest that
resource management is only possible on a proper subset of archs. Since
this is not the case wouldn't it be better to source the bc Kconfig from
an arch-independent Kconfig (init/Kconfig for example)?

> > PS: Please keep ckrm-tech on Cc: please.

Also, thank you for CC'ing CKRM-Tech with your earlier posting of these
patches.

> Sorry, it is very hard to track emails coming from authors and 3 mailing lists.

Yes, it can be difficult to keep track of all the email authors.

> Better tell me the interested people emails.

 CC'ing only the known-interested people wouldn't be better. If anything
I think it's harder for everyone than simply CC'ing a relevant mailing
list like LKML and CKRM-Tech in this case.

Cheers,
 -Matt Helsley

Subject: Re: [PATCH 5/6] BC: kernel memory accounting (core)
Posted by dev on Fri, 25 Aug 2006 10:06:45 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 08/23, Kirill Korotaev wrote:
>
>>+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();
>>+ if (bc == NULL)
>>+ return 0;
>
>
> Is it possible to .exec_bc == NULL ?
>
> If yes, why do we need init_bc? We can do 'set_exec_bc(NULL)' in __do_IRQ()
> instead.
no, exec_bc can't be NULL. thanks for catching old hunks which historically exist
due to old times when host system was not accounted (bc was NULL).

Thanks,
Kirill

Subject: Re: [PATCH 2/6] BC: beancounters core (API)
Posted by dev on Fri, 25 Aug 2006 10:51:18 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:
> On Thu, 24 Aug 2006 16:06:11 +0400
> Kirill Korotaev <dev@sw.ru> wrote:
>
>
>>>>>+#define bc_charge_locked(bc, r, v, s)   (0)
>>>>
>>>>>+#define bc_charge(bc, r, v)   (0)
>>>
>>>akpm:/home/akpm> cat t.c
>>>void foo(void)

>>>{
>>> (0);
>>>}
>>>akpm:/home/akpm> gcc -c -Wall t.c
>>>t.c: In function 'foo':
>>>t.c:4: warning: statement with no effect
>>
>>these functions return value should always be checked (!).
>
>
> We have __must_check for that.
>
>
>>i.e. it is never called like:
>> ub_charge(bc, r, v);
>
>
> Also...
>
> if (bc_charge(tpyo, undefined_variable, syntax_error))
>
> will happily compile if !CONFIG_BEANCOUNTER.
>
> Turning these stubs into static inline __must_check functions fixes all this.

ok. will replace all empty stubs with inlines (with __must_check where appropriate)

Thanks,
Kirill

Subject: Re: [PATCH 4/6] BC: user interface (syscalls)
Posted by dev on Fri, 25 Aug 2006 10:54:18 GMT
View Forum Message <> Reply to Message

Alexey Dobriyan wrote:
> On Thu, Aug 24, 2006 at 12:04:16PM +0100, Alan Cox wrote:
>
>>Ar Mer, 2006-08-23 am 21:35 -0700, ysgrifennodd Andrew Morton:
>>
>>>>Its a uid_t because of setluid() and twenty odd years of existing unix
>>>>practice.
>>>>
>>>
>>>I don't understand. This number is an identifier for an accounting
>>>container, which was somehow dreamed up by userspace.
>>
>>Which happens to be a uid_t. It could easily be anyother_t of itself and

>>you can create a container_id_t or whatever. It is just a number.
>>
>>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. So we can certainly do something like
>>
>>typedef uid_t container_id_t;
>
>
> What about cid_t? Google mentions cid_t was used in HP-UX specific IPC (only if
> _INCLUDE_HPUX_SOURCE is defined).
bcid_t?

Thanks,
Kirill

Subject: Re: [PATCH 1/6] BC: kconfig
Posted by dev on Fri, 25 Aug 2006 11:27:40 GMT
View Forum Message <> Reply to Message

Matt Helsley wrote:
> On Wed, 2006-08-23 at 15:04 -0700, Dave Hansen wrote:
>
>>On Wed, 2006-08-23 at 15:01 +0400, Kirill Korotaev wrote:
>>
>>>--- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
>>>+++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
>>>@@ -432,3 +432,5 @@ source "security/Kconfig"
>>> source "crypto/Kconfig"
>>>
>>> source "lib/Kconfig"
>>>+
>>>+source "kernel/bc/Kconfig"
>>
>>...
>>
>>>--- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
>>>+++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
>>>@@ -432,3 +432,5 @@ source "security/Kconfig"
>>> source "crypto/Kconfig"
>>>

>>> source "lib/Kconfig"
>>>+
>>>+source "kernel/bc/Kconfig"
>>
>>Is it just me, or do these patches look a little funky? Looks like it
>>is trying to patch the same thing into the same file, twice. Also, the
>>patches look to be -p0 instead of -p1.
>
>
> They do appear to be -p0
it is -p1. patches are generated with gendiff and ./ in names is for -p1

> They aren't adding the same thing twice to the same file. This patch
> makes different arches source the same Kconfig.
>
> I seem to recall Chandra suggested that instead of doing it this way it
> would be more appropriate to add the source line to init/Kconfig because
> it's more central and arch-independent. I tend to agree.
agreed. init/Kconfig looks like a good place for including
kernel/bc/Kconfig

Kirill

Subject: Re: [PATCH 1/6] BC: kconfig
Posted by dev on Fri, 25 Aug 2006 11:31:28 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Wed, 2006-08-23 at 15:01 +0400, Kirill Korotaev wrote:
>
>>--- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
>>+++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
>>@@ -432,3 +432,5 @@ source "security/Kconfig"
>> source "crypto/Kconfig"
>>
>> source "lib/Kconfig"
>>+
>>+source "kernel/bc/Kconfig"
>
> ...
>
>>--- ./arch/sparc64/Kconfig.arkcfg 2006-07-17 17:01:11.000000000 +0400
>>+++ ./arch/sparc64/Kconfig 2006-08-10 17:56:36.000000000 +0400
>>@@ -432,3 +432,5 @@ source "security/Kconfig"
>> source "crypto/Kconfig"
>>
>> source "lib/Kconfig"

>>+
>>+source "kernel/bc/Kconfig"
>
>
> Is it just me, or do these patches look a little funky? Looks like it
> is trying to patch the same thing into the same file, twice. Also, the
> patches look to be -p0 instead of -p1.
>
> I'm having a few problems applying them.
Oh, it's my fault. I pasted text twice :/

Kirill

Subject: Re: [PATCH] BC: resource beancounters (v2)
Posted by dev on Fri, 25 Aug 2006 11:47:28 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:
>>As the first step we want to propose for discussion
>>the most complicated parts of resource management:
>>kernel memory and virtual memory.
>
>
> The patches look reasonable to me - mergeable after updating them for
> today's batch of review commentlets.
sure. will do updates as long as there are reasonable comments.

> I have two high-level problems though.
>
> a) I don't yet have a sense of whether this implementation
>   is appropriate/sufficient for the various other
>   applications which people are working on.
>
>   If the general shape is OK and we think this
>   implementation can be grown into one which everyone can
>   use then fine.
>
> And...
>
>
>>The patch set to be sent provides core for BC and
>>management of kernel memory only. Virtual memory
>>management will be sent in a couple of days.
>
>
> We need to go over this work before we can commit to the BC
> core. Last time I looked at the VM accounting patch it

> seemed rather unpleasing from a maintainability POV.
hmmm... in which regard?

> And, if I understand it correctly, the only response to a job
> going over its VM limits is to kill it, rather than trimming
> it. Which sounds like a big problem?
No, UBC virtual memory management refuses occur on mmap()'s.
Andrey Savochkin wrote already a brief summary on vm resource management:

---------- cut ----------
The task of limiting a container to 4.5GB of memory bottles down to the
question: what to do when the container starts to use more than assigned
4.5GB of memory?

At this moment there are only 3 viable alternatives.

A) Have separate memory management for each container,
   with separate buddy allocator, lru lists, page replacement mechanism.
   That implies a considerable overhead, and the main challenge there
   is sharing of pages between these separate memory managers.

B) Return errors on extension of mappings, but not on page faults, where
   memory is actually consumed.
   In this case it makes sense to take into account not only the size of used
   memory, but the size of created mappings as well.
   This is approximately what "privvmpages" accounting/limiting provides in
   UBC.

C) Rely on OOM killer.
   This is a fall-back method in UBC, for the case "privvmpages" limits
   still leave the possibility to overload the system.

It would be nice, indeed, to invent something new.
The ideal mechanism would
 - slow down the container over-using memory, to signal the user that
   he is over his limits,
 - at the same time this slowdown shouldn't lead to the increase of memory
   usage: for example, a simple slowdown of apache web server would lead
   to the growth of the number of serving children and consumption of more
   memory while showing worse performance,
 - and, at the same time, it shouldn't penalize the rest of the system from
   the performance point of view...
May be this can be achieved via carefully tuned swapout mechanism together
with disk bandwidth management capable of tracking asynchronous write
requests, may be something else is required.
It's really a big challenge.

Meanwhile, I guess we can only make small steps in improving Linux resource

management features for this moment.
---------- cut ----------

Thanks,
Kirill

Subject: Re: [PATCH] BC: resource beancounters (v2)
Posted by Andrew Morton on Fri, 25 Aug 2006 14:30:03 GMT
View Forum Message <> Reply to Message

On Fri, 25 Aug 2006 15:49:15 +0400
Kirill Korotaev <dev@sw.ru> wrote:

> > We need to go over this work before we can commit to the BC
> > core. Last time I looked at the VM accounting patch it
> > seemed rather unpleasing from a maintainability POV.
> hmmm... in which regard?

Little changes all over the MM code which might get accidentally broken.

> > And, if I understand it correctly, the only response to a job
> > going over its VM limits is to kill it, rather than trimming
> > it. Which sounds like a big problem?
> No, UBC virtual memory management refuses occur on mmap()'s.

That's worse, isn't it? Firstly it rules out big sparse mappings and secondly

 mmap_and_use(80% of container size)
 fork_and_immediately_exec(/bin/true)

will fail at the fork?

> Andrey Savochkin wrote already a brief summary on vm resource management:
>
> ---------- cut ----------
> The task of limiting a container to 4.5GB of memory bottles down to the
> question: what to do when the container starts to use more than assigned
> 4.5GB of memory?
>
> At this moment there are only 3 viable alternatives.
>
> A) Have separate memory management for each container,
>   with separate buddy allocator, lru lists, page replacement mechanism.
>   That implies a considerable overhead, and the main challenge there
>   is sharing of pages between these separate memory managers.
>

> B) Return errors on extension of mappings, but not on page faults, where
> memory is actually consumed.
> In this case it makes sense to take into account not only the size of used
> memory, but the size of created mappings as well.
> This is approximately what "privvmpages" accounting/limiting provides in
> UBC.
>
> C) Rely on OOM killer.
> This is a fall-back method in UBC, for the case "privvmpages" limits
> still leave the possibility to overload the system.
>

D) Virtual scan of mm's in the over-limit container

E) Modify existing physical scanner to be able to skip pages which
 belong to not-over-limit containers.

F) Something else ;)

Subject: Re: [PATCH] BC: resource beancounters (v2)
Posted by Andi Kleen on Fri, 25 Aug 2006 14:48:00 GMT
View Forum Message <> Reply to Message

> D) Virtual scan of mm's in the over-limit container
>
> E) Modify existing physical scanner to be able to skip pages which
> belong to not-over-limit containers.

The same applies to dentries/inodes too, doesn't it? But their
scan is already virtual so their LRUs could be just split into
a per container list (but then didn't Christoph L. plan to
rewrite that code anyways?)

For memory my guess would be that (E) would be easier than (D) for user/file
memory though less efficient.

-Andi

Subject: Re: [PATCH] BC: resource beancounters (v2)
Posted by Nick Piggin on Fri, 25 Aug 2006 15:14:03 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:
> On Fri, 25 Aug 2006 15:49:15 +0400
> Kirill Korotaev <dev@sw.ru> wrote:

>
>
>>>We need to go over this work before we can commit to the BC
>>>core. Last time I looked at the VM accounting patch it
>>>seemed rather unpleasing from a maintainability POV.
>>
>>hmmm... in which regard?
>
>
> Little changes all over the MM code which might get accidentally broken.

I still think doing simple accounting per-page would be a better way to
go than trying to pin down all "user allocatable" kernel allocations.
And would require all of about 2 hooks in the page allocator. And would
track *actual* RAM allocated by that container.

Can we continue that discussion (ie. why it isn't good enough). Last I
was told it is not perfect and can be unfair... sounds like it fits the
semantics perfectly ;)

--
SUSE Labs, Novell Inc.

Subject: Re: [PATCH] BC: resource beancounters (v2)
Posted by Alan Cox on Fri, 25 Aug 2006 15:36:08 GMT
View Forum Message <> Reply to Message

Ar Sad, 2006-08-26 am 01:14 +1000, ysgrifennodd Nick Piggin:
> I still think doing simple accounting per-page would be a better way to
> go than trying to pin down all "user allocatable" kernel allocations.
> And would require all of about 2 hooks in the page allocator. And would
> track *actual* RAM allocated by that container.

You have a variety of kernel objects you want to worry about and they
have very differing properties.

Some are basically shared resources - page cache, dentries, inodes, etc
and can be balanced pretty well by the kernel (ok the dentries are a bit
of a problem right now). Others are very specific "owned" resources -
like file handles, sockets and vmas.

Tracking actual RAM use by container/user/.. isn't actually that
interesting. It's also inconveniently sub page granularity.

Its a whole seperate question whether you want a separate bean counter
limit for sockets, file handles, vmas etc.

Alan

Subject: Re: BC: resource beancounters (v2)
Posted by Andrey Savochkin on Fri, 25 Aug 2006 16:30:26 GMT
View Forum Message <> Reply to Message

On Fri, Aug 25, 2006 at 07:30:03AM -0700, Andrew Morton wrote:
> On Fri, 25 Aug 2006 15:49:15 +0400
> Kirill Korotaev <dev@sw.ru> wrote:
>
> > Andrey Savochkin wrote already a brief summary on vm resource management:
> >
> > ---------- cut ----------
> > The task of limiting a container to 4.5GB of memory bottles down to the
> > question: what to do when the container starts to use more than assigned
> > 4.5GB of memory?
> >
> > At this moment there are only 3 viable alternatives.
> >
> > A) Have separate memory management for each container,
> >   with separate buddy allocator, lru lists, page replacement mechanism.
> >   That implies a considerable overhead, and the main challenge there
> >   is sharing of pages between these separate memory managers.
> >
> > B) Return errors on extension of mappings, but not on page faults, where
> >   memory is actually consumed.
> >   In this case it makes sense to take into account not only the size of used
> >   memory, but the size of created mappings as well.
> >   This is approximately what "privvmpages" accounting/limiting provides in
> >   UBC.
> >
> > C) Rely on OOM killer.
> >   This is a fall-back method in UBC, for the case "privvmpages" limits
> >   still leave the possibility to overload the system.
> >
>
> D) Virtual scan of mm's in the over-limit container
>
> E) Modify existing physical scanner to be able to skip pages which
>   belong to not-over-limit containers.

I've actually tried (E), but it didn't work as I wished.

It didn't handle well shared pages.
Then, in my experiments such modified scanner was unable to regulate
quality-of-service. When I ran 2 over-the-limit containers, they worked

equally slow regardless of their limits and work set size.
That is, I didn't observe a smooth transition "under limit, maximum
performance" to "slightly over limit, a bit reduced performance" to
"significantly over limit, poor performance". Neither did I see any fairness
in how containers got penalized for exceeding their limits.

My explanation of what I observed is that
- since filesystem caches play a huge role in performance, page scanner will
  be very limited in controlling container's performance if caches
  stay shared between containers,
- in the absence of decent disk I/O manager, stalls due to swapin/swapout
  are more influenced by disk subsystem than by page scanner policy.

So in fact modified page scanner provides control over memory usage only as
"stay under limits or die", and doesn't show many advantages over (B) or (C).
At the same time, skipping pages visibly penalizes "good citizens", not only
in disk bandwidth but in CPU overhead as well.

So I settled for (A)-(C) for now.
But it certainly would be interesting to hear if someone else makes such
experiments.

Best regards

Andrey

Subject: Re: BC: resource beancounters (v2)
Posted by Andrew Morton on Fri, 25 Aug 2006 17:50:47 GMT
View Forum Message <> Reply to Message

On Fri, 25 Aug 2006 20:30:26 +0400
Andrey Savochkin <saw@sw.ru> wrote:

> On Fri, Aug 25, 2006 at 07:30:03AM -0700, Andrew Morton wrote:
> >
> > D) Virtual scan of mm's in the over-limit container
> >
> > E) Modify existing physical scanner to be able to skip pages which
> >   belong to not-over-limit containers.
>
> I've actually tried (E), but it didn't work as I wished.
>
> It didn't handle well shared pages.
> Then, in my experiments such modified scanner was unable to regulate
> quality-of-service. When I ran 2 over-the-limit containers, they worked
> equally slow regardless of their limits and work set size.
> That is, I didn't observe a smooth transition "under limit, maximum
> performance" to "slightly over limit, a bit reduced performance" to

> "significantly over limit, poor performance". Neither did I see any fairness
> in how containers got penalized for exceeding their limits.
>
> My explanation of what I observed is that
> - since filesystem caches play a huge role in performance, page scanner will
> be very limited in controlling container's performance if caches
> stay shared between containers,
> - in the absence of decent disk I/O manager, stalls due to swapin/swapout
> are more influenced by disk subsystem than by page scanner policy.
> So in fact modified page scanner provides control over memory usage only as
> "stay under limits or die", and doesn't show many advantages over (B) or (C).
> At the same time, skipping pages visibly penalizes "good citizens", not only
> in disk bandwidth but in CPU overhead as well.
>
> So I settled for (A)-(C) for now.
> But it certainly would be interesting to hear if someone else makes such
> experiments.
>

Makes sense. If one is looking for good machine partitioning then a shared disk is obviously a great contention point. To address that we'd need to be able to say "container A swaps to /dev/sda1 and container B swaps to /dev/sdb1". But the swap system at present can't do that.

Subject: Re: BC: resource beancounters (v2)
Posted by Chandra Seetharaman on Fri, 25 Aug 2006 19:00:05 GMT
View Forum Message <> Reply to Message

Have you seen/tried the memory controller in CKRM/Resource Groups ?
http://sourceforge.net/projects/ckrm

It maintains a per resource group LRU lists and also maintains a list of over-guarantee groups (with ordering based on where they are in their guarantee-limit scale). So, when a reclaim needs to happen, pages are first freed from a group that is way over its limit, and then the next one and so on.

Few things that it does that are not good:
- doesn't account shared pages accurately
- moves all pages from a task when the task moves to a different group
- totally new reclamation path

regards,

chandra
On Fri, 2006-08-25 at 20:30 +0400, Andrey Savochkin wrote:
> On Fri, Aug 25, 2006 at 07:30:03AM -0700, Andrew Morton wrote:

> > On Fri, 25 Aug 2006 15:49:15 +0400
> > Kirill Korotaev <dev@sw.ru> wrote:
> >
> > > Andrey Savochkin wrote already a brief summary on vm resource management:
> > >
> > > ---------- cut ----------
> > > The task of limiting a container to 4.5GB of memory bottles down to the
> > > question: what to do when the container starts to use more than assigned
> > > 4.5GB of memory?
> > >
> > > At this moment there are only 3 viable alternatives.
> > >
> > > A) Have separate memory management for each container,
> > >   with separate buddy allocator, lru lists, page replacement mechanism.
> > >   That implies a considerable overhead, and the main challenge there
> > >   is sharing of pages between these separate memory managers.
> > >
> > > B) Return errors on extension of mappings, but not on page faults, where
> > >   memory is actually consumed.
> > >   In this case it makes sense to take into account not only the size of used
> > >   memory, but the size of created mappings as well.
> > >   This is approximately what "privvmpages" accounting/limiting provides in
> > >   UBC.
> > >
> > > C) Rely on OOM killer.
> > >   This is a fall-back method in UBC, for the case "privvmpages" limits
> > >   still leave the possibility to overload the system.
> > >
> >
> > D) Virtual scan of mm's in the over-limit container
> >
> > E) Modify existing physical scanner to be able to skip pages which
> >   belong to not-over-limit containers.
>
> I've actually tried (E), but it didn't work as I wished.
>
> It didn't handle well shared pages.
> Then, in my experiments such modified scanner was unable to regulate
> quality-of-service. When I ran 2 over-the-limit containers, they worked
> equally slow regardless of their limits and work set size.
> That is, I didn't observe a smooth transition "under limit, maximum
> performance" to "slightly over limit, a bit reduced performance" to
> "significantly over limit, poor performance". Neither did I see any fairness
> in how containers got penalized for exceeding their limits.
>
> My explanation of what I observed is that
> - since filesystem caches play a huge role in performance, page scanner will
>   be very limited in controlling container's performance if caches

> stay shared between containers,
> - in the absence of decent disk I/O manager, stalls due to swapin/swapout
> are more influenced by disk subsystem than by page scanner policy.
> So in fact modified page scanner provides control over memory usage only as
> "stay under limits or die", and doesn't show many advantages over (B) or (C).
> At the same time, skipping pages visibly penalizes "good citizens", not only
> in disk bandwidth but in CPU overhead as well.
>
> So I settled for (A)-(C) for now.
> But it certainly would be interesting to hear if someone else makes such
> experiments.
>
> Best regards
>
> Andrey
--

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

Subject: Re: BC: resource beancounters (v2)
Posted by Rohit Seth on Sat, 26 Aug 2006 02:15:52 GMT
View Forum Message <> Reply to Message

On Fri, 2006-08-25 at 20:30 +0400, Andrey Savochkin wrote:
> On Fri, Aug 25, 2006 at 07:30:03AM -0700, Andrew Morton wrote:
> > On Fri, 25 Aug 2006 15:49:15 +0400
> > Kirill Korotaev <dev@sw.ru> wrote:
> >
> > > Andrey Savochkin wrote already a brief summary on vm resource management:
> > >
> > > ---------- cut ----------
> > > The task of limiting a container to 4.5GB of memory bottles down to the
> > > question: what to do when the container starts to use more than assigned
> > > 4.5GB of memory?
> > >
> > > At this moment there are only 3 viable alternatives.
> > >
> > > A) Have separate memory management for each container,
> > > with separate buddy allocator, lru lists, page replacement mechanism.
> > > That implies a considerable overhead, and the main challenge there
> > > is sharing of pages between these separate memory managers.
> > >

Yes, sharing of pages across different containers/managers will be a

problem. Why not just disallow that scenario (that is what fake nodes
proposal would also end up doing).

> > > B) Return errors on extension of mappings, but not on page faults, where
> > >   memory is actually consumed.
> > >   In this case it makes sense to take into account not only the size of used
> > >   memory, but the size of created mappings as well.
> > >   This is approximately what "privvmpages" accounting/limiting provides in
> > >   UBC.
> > >

Keeping a tab on all the virtual mappings in a container must also be
troublesome. And IMO is not the right way to go...this is even a
stricter version of overcommit_memory...right?
>
> > > C) Rely on OOM killer.
> > >   This is a fall-back method in UBC, for the case "privvmpages" limits
> > >   still leave the possibility to overload the system.
> > >
> >
> > D) Virtual scan of mm's in the over-limit container
> >

This seems like an interesting choice. If we can quickly inactivate
some pages belonging to tasks in over_the_limit container.

> > E) Modify existing physical scanner to be able to skip pages which
> >   belong to not-over-limit containers.
>
> I've actually tried (E), but it didn't work as I wished.
>
> It didn't handle well shared pages.
> Then, in my experiments such modified scanner was unable to regulate
> quality-of-service. When I ran 2 over-the-limit containers, they worked
> equally slow regardless of their limits and work set size.
> That is, I didn't observe a smooth transition "under limit, maximum
> performance" to "slightly over limit, a bit reduced performance" to
> "significantly over limit, poor performance". Neither did I see any fairness
> in how containers got penalized for exceeding their limits.
>

That sure is an interesting observation though I think it really depends
on if you are doing the same amount of work when counts have just gone
above the limits to the point where they are way over the limit.

> My explanation of what I observed is that
> - since filesystem caches play a huge role in performance, page scanner will
>   be very limited in controlling container's performance if caches

> stay shared between containers,

Yeah, if a page is shared between containers then you can end up doing
nothing useful. And that is where containers dedicated to individual
filesystem could be useful.

> - in the absence of decent disk I/O manager, stalls due to swapin/swapout
> are more influenced by disk subsystem than by page scanner policy.
> So in fact modified page scanner provides control over memory usage only as
> "stay under limits or die", and doesn't show many advantages over (B) or (C).
> At the same time, skipping pages visibly penalizes "good citizens", not only
> in disk bandwidth but in CPU overhead as well.
>

Sure that CPU, disk and other variables will kick in when you start
swapping. But then apps are expected to suffer when gone over limit.
The drawback is the apps that are not hit the limit will also suffer,
but then that is where extra controllers like CPU will kick in.

Maybe, we have a flag for each container indicating whether the tasks
belonging to that container should be killed immediately or they are
okay to run with lower performance as far as they can.

-rohit

Subject: Re: [PATCH] BC: resource beancounters (v2)
Posted by Nick Piggin on Sat, 26 Aug 2006 03:55:43 GMT
View Forum Message <> Reply to Message

Alan Cox wrote:
> Ar Sad, 2006-08-26 am 01:14 +1000, ysgrifennodd Nick Piggin:
>
>>I still think doing simple accounting per-page would be a better way to
>>go than trying to pin down all "user allocatable" kernel allocations.
>>And would require all of about 2 hooks in the page allocator. And would
>>track *actual* RAM allocated by that container.
>
>
> You have a variety of kernel objects you want to worry about and they
> have very differing properties.
>
> Some are basically shared resources - page cache, dentries, inodes, etc
> and can be balanced pretty well by the kernel (ok the dentries are a bit
> of a problem right now). Others are very specific "owned" resources -
> like file handles, sockets and vmas.

That's true (OTOH I'd argue it would still be very useful for things

like pagecache, so one container can't start a couple of 'dd' loops
and turn everyone else to crap). And while the sharing may not be
exactly captured, statistically things should balance over time.

So I'm not arguing about _also_ accounting resources that are limited
in other ways (than just the RAM they consume).

But as a DoS protection measure on RAM usage, trying to account all
kernel allocations that are user triggerable just sounds hard to
maintain, holey, ugly, invsive (and not perfect either -- in fact it
still isn't clear to me that it is any better than my proposal).

>
> Tracking actual RAM use by container/user/.. isn't actually that
> interesting. It's also inconveniently sub page granularity.

If it isn't interesting, then I don't think we want it (at least, until
someone does get an interest in it).

>
> Its a whole seperate question whether you want a separate bean counter
> limit for sockets, file handles, vmas etc.

Yeah that's fair enough. We obviously want to avoid exposing limits on
things that it doesn't make sense to limit, or that is a kernel
implementation detail as much as possible.

eg. so I would be happy to limit virtual address, less happy to limit
vmas alone (unless that is in the context of accounting their RAM usage
or their implied vaddr charge).

--
SUSE Labs, Novell Inc.

Subject: Re: BC: resource beancounters (v2)
Posted by Alan Cox on Sat, 26 Aug 2006 16:16:02 GMT
View Forum Message <> Reply to Message

Ar Gwe, 2006-08-25 am 19:15 -0700, ysgrifennodd Rohit Seth:
> Yes, sharing of pages across different containers/managers will be a
> problem. Why not just disallow that scenario (that is what fake nodes
> proposal would also end up doing).

Because it destroys the entire point of using containers instead of
something like Xen - which is sharing. Also at the point I am using
beancounters per user I don't want glibc per use, libX11 per use glib

per use gtk per user etc..

Subject: Re: [PATCH] BC: resource beancounters (v2)
Posted by dev on Mon, 28 Aug 2006 08:27:16 GMT
View Forum Message <> Reply to Message

Andi Kleen wrote:
>>D) Virtual scan of mm's in the over-limit container
>>
>>E) Modify existing physical scanner to be able to skip pages which
>> belong to not-over-limit containers.
>
>
> The same applies to dentries/inodes too, doesn't it? But their
> scan is already virtual so their LRUs could be just split into
> a per container list (but then didn't Christoph L. plan to
> rewrite that code anyways?)
how do you propose to handle shared files in this case?

> For memory my guess would be that (E) would be easier than (D) for user/file
> memory though less efficient.

Kirill

Subject: Re: BC: resource beancounters (v2)
Posted by Rohit Seth on Mon, 28 Aug 2006 16:48:41 GMT
View Forum Message <> Reply to Message

On Sat, 2006-08-26 at 17:37 +0100, Alan Cox wrote:
> Ar Gwe, 2006-08-25 am 19:15 -0700, ysgrifennodd Rohit Seth:
> > Yes, sharing of pages across different containers/managers will be a
> > problem. Why not just disallow that scenario (that is what fake nodes
> > proposal would also end up doing).
>
> Because it destroys the entire point of using containers instead of
> something like Xen - which is sharing. Also at the point I am using
> beancounters per user I don't want glibc per use, libX11 per use glib
> per use gtk per user etc..
>
>

I'm not saying per use glibc etc. That will indeed be useless and bring
it to virtualization world. Just like fake node, one should be allowed
to use pages that are already in (for example) page cache- so that you
don't end up duplicating all shared stuff. But as far as charging is

concerned, charge it to container who either got the page in page cache
OR if FS based semantics exist then charge it to the container where the
file belongs. What I was suggesting is to not charge a page to
different counters.

-rohit

Subject: Re: Re: BC: resource beancounters (v2)
Posted by kir on Mon, 28 Aug 2006 17:40:18 GMT
View Forum Message <> Reply to Message

Rohit Seth wrote:
> On Sat, 2006-08-26 at 17:37 +0100, Alan Cox wrote:
>
>> Ar Gwe, 2006-08-25 am 19:15 -0700, ysgrifennodd Rohit Seth:
>>
>>> Yes, sharing of pages across different containers/managers will be a
>>> problem. Why not just disallow that scenario (that is what fake nodes
>>> proposal would also end up doing).
>>>
>> Because it destroys the entire point of using containers instead of
>> something like Xen - which is sharing. Also at the point I am using
>> beancounters per user I don't want glibc per use, libX11 per use glib
>> per use gtk per user etc..
>>
>>
>>
>
> I'm not saying per use glibc etc. That will indeed be useless and bring
> it to virtualization world. Just like fake node, one should be allowed
> to use pages that are already in (for example) page cache- so that you
> don't end up duplicating all shared stuff. But as far as charging is
> concerned, charge it to container who either got the page in page cache
> OR if FS based semantics exist then charge it to the container where the
> file belongs. What I was suggesting is to not charge a page to
> different counters.
>

Consider the following simple scenario: there are 50 containers
(numbered, say, 1 to 50) all sharing a single installation of Fedora
Core 5. They all run sshd, apache, syslogd, crond and some other stuff
like that. This is actually quite a real scenario.

In the world that you propose the container which was unlucky to start
first (probably the one with ID of either 1 or 50) will be charged for
all the memory, and all the
others will have most of their memory for free. And in such a world

per-container memory accounting or limiting is just not possible.

Subject: Re: Re: BC: resource beancounters (v2)
Posted by Rohit Seth on Mon, 28 Aug 2006 22:28:09 GMT
View Forum Message <> Reply to Message

On Mon, 2006-08-28 at 21:41 +0400, Kir Kolyshkin wrote:
> Rohit Seth wrote:
> >
> > I'm not saying per use glibc etc. That will indeed be useless and bring
> > it to virtualization world. Just like fake node, one should be allowed
> > to use pages that are already in (for example) page cache- so that you
> > don't end up duplicating all shared stuff. But as far as charging is
> > concerned, charge it to container who either got the page in page cache
> > OR if FS based semantics exist then charge it to the container where the
> > file belongs. What I was suggesting is to not charge a page to
> > different counters.
> >
>
> Consider the following simple scenario: there are 50 containers
> (numbered, say, 1 to 50) all sharing a single installation of Fedora
> Core 5. They all run sshd, apache, syslogd, crond and some other stuff
> like that. This is actually quite a real scenario.
>
> In the world that you propose the container which was unlucky to start
> first (probably the one with ID of either 1 or 50) will be charged for
> all the memory, and all the
> others will have most of their memory for free. And in such a world
> per-container memory accounting or limiting is just not possible.

If you are only having task based accounting then yes the first
container using a page will be charged. And when it hit its limit then
it will inactivate some of the pages. If some other container now uses
the same page (that got inactivated) again then this next container will
be charged for that page.

Though if we have file/directory based accounting then shared pages
belonging to /usr/lib or /usr/bin can go to a common container.

-rohit

Subject: Re: [PATCH 6/6] BC: kernel memory accounting (marks)
Posted by dev on Tue, 29 Aug 2006 09:52:54 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> I'm still a bit concerned about if we actually need the 'struct page'
> pointer. I've gone through all of the users, and I'm not sure that I
> see any that _require_ having a pointer in 'struct page'. I think it
> will take some rework, especially with the pagetables, but it should be
> quite doable.
don't worry:
1. we will introduce a separate patch moving this pointer
  into mirroring array
2. this pointer is still required for _user_ pages tracking,
  that's why I don't follow your suggestion right now...

> vmalloc:
> Store in vm_struct
> fd_set_bits:
> poll_get:
> mount hashtable:
> Don't need alignment. use the slab?
> pagetables:
> either store in an extra field of 'struct page', or use the
> mm's. mm should always be available when alloc/freeing a
> pagetable page
>
> Did I miss any?
flocks, pipe buffers, task_struct, sighand, signal, vmas,
posix timers, uid_cache, shmem dirs,

Thanks,
Kirill

Subject: Re: Re: BC: resource beancounters (v2)
Posted by Alan Cox on Tue, 29 Aug 2006 10:15:35 GMT
View Forum Message <> Reply to Message

Ar Llu, 2006-08-28 am 15:28 -0700, ysgrifennodd Rohit Seth:
> Though if we have file/directory based accounting then shared pages
> belonging to /usr/lib or /usr/bin can go to a common container.

So that one user can map all the spare libraries and config files and
DoS the system by preventing people from accessing the libraries they do
need ?

Subject: Re: [PATCH 6/6] BC: kernel memory accounting (marks)
Posted by dev on Tue, 29 Aug 2006 14:34:31 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Wed, 2006-08-23 at 15:08 +0400, Kirill Korotaev wrote:
>
>> 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 +++--
>
>
> Do you think we need to cover a few more architectures before
> considering merging this, or should we just fix them up as we need them?
I think doing a part of job is usually bad as it never gets fixed fully then :/

> I'm working on a patch to unify as many of the alloc_thread_info()
> functions as I can. That should at least give you one place to modify
> and track the thread_info allocations. I've only compiled for x86_64
> and i386, but I'm working on more. A preliminary version is attached.
Oh, I think such code unification is nice!
Would be perferct if all of them could be merged to some generic
function. Please, keep me on CC!

Thanks,
Kirill

Subject: Re: [PATCH] BC: resource beancounters (v2)
Posted by dev on Tue, 29 Aug 2006 15:33:04 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:
> On Fri, 25 Aug 2006 15:49:15 +0400
> Kirill Korotaev <dev@sw.ru> wrote:
>
>
>>>We need to go over this work before we can commit to the BC
>>>core. Last time I looked at the VM accounting patch it
>>>seemed rather unpleasing from a maintainability POV.
>>
>>hmmm... in which regard?
>
>
> Little changes all over the MM code which might get accidentally broken.
>
>
>>>And, if I understand it correctly, the only response to a job
>>>going over its VM limits is to kill it, rather than trimming
>>>it. Which sounds like a big problem?
>>

>>No, UBC virtual memory management refuses occur on mmap()'s.
>
>
> That's worse, isn't it? Firstly it rules out big sparse mappings and secondly
1) if mappings are private then yes, you can not mmap too much. This is logical,
since this whole mappings are potentially allocatable and there is no way to control
it later except for SIGKILL.
2) if mappings are shared file mappings (shmem is handled in similar way) then
  you can mmap as much as you want, since these pages can be reclaimed.

> mmap_and_use(80% of container size)
> fork_and_immediately_exec(/bin/true)
>
> will fail at the fork?
yes, it will fail on fork() or exec() in case of much private (1) mappings.
fail on fork() and exec() is much friendlier then SIGKILL, don't you think so?

private mappings parameter which is limited by UBC is a kind of upper estimation
of container RSS. From our experience such an estimation is ~ 5-20% higher then
a real physical memory used (with real-life applications).

>>Andrey Savochkin wrote already a brief summary on vm resource management:
>>
>>------------- cut ----------------
>>The task of limiting a container to 4.5GB of memory bottles down to the
>>question: what to do when the container starts to use more than assigned
>>4.5GB of memory?
>>
>>At this moment there are only 3 viable alternatives.
>>
>>A) Have separate memory management for each container,
>>   with separate buddy allocator, lru lists, page replacement mechanism.
>>   That implies a considerable overhead, and the main challenge there
>>   is sharing of pages between these separate memory managers.
>>
>>B) Return errors on extension of mappings, but not on page faults, where
>>   memory is actually consumed.
>>   In this case it makes sense to take into account not only the size of used
>>   memory, but the size of created mappings as well.
>>   This is approximately what "privvmpages" accounting/limiting provides in
>>   UBC.
>>
>>C) Rely on OOM killer.
>>   This is a fall-back method in UBC, for the case "privvmpages" limits
>>   still leave the possibility to overload the system.
>>
>
>

> D) Virtual scan of mm's in the over-limit container
>
> E) Modify existing physical scanner to be able to skip pages which
> belong to not-over-limit containers.
>
> F) Something else ;)
We fully agree that other possible algorithms can and should exist.
My idea only is that any of them would need accounting anyway
(which is the most part of beancounters).
Throtling, modified scanners etc. can be implemented as a separate
BC parameters. Thus, an administrator will be able to select
which policy should be applied to the container which is near its limit.

So the patches I'm trying to send are a step-by-step accounting of all
the resources and their simple limitations. More comprehensive limitation
policy will be built on top of it later.

BTW, UBC page beancounters allow to distinguish pages used by only one
container and pages which are shared. So scanner can try to reclaim
container private pages first, thus not influencing other containers.

Thanks,
Kirill

Subject: Re: [PATCH 6/6] BC: kernel memory accounting (marks)
Posted by Dave Hansen on Tue, 29 Aug 2006 15:48:45 GMT
View Forum Message <> Reply to Message

On Tue, 2006-08-29 at 13:52 +0400, Kirill Korotaev wrote:
> 1. we will introduce a separate patch moving this pointer
> into mirroring array
> 2. this pointer is still required for _user_ pages tracking,
> that's why I don't follow your suggestion right now...

You hadn't mentioned that part. ;)

I guess we'll wait for the user patches before these can go any further.

-- Dave

Subject: Re: [PATCH 6/6] BC: kernel memory accounting (marks)
Posted by dev on Tue, 29 Aug 2006 15:53:09 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:

> On Tue, 2006-08-29 at 13:52 +0400, Kirill Korotaev wrote:
>
>>1. we will introduce a separate patch moving this pointer
>> into mirroring array
>>2. this pointer is still required for _user_ pages tracking,
>> that's why I don't follow your suggestion right now...
>
>
> You hadn't mentioned that part. ;)
I was crying about it 2 or 3 times :)
But no suprise that you hadn't notice that due to too many emails on BC.

> I guess we'll wait for the user patches before these can go any further.
:))) are you the one to decide? :) then lets drink some beer :)))

Thanks,
Kirill

Subject: Re: [PATCH] BC: resource beancounters (v2)
Posted by Balbir Singh on Tue, 29 Aug 2006 17:08:45 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
>>> ------------ cut ----------------
>>> The task of limiting a container to 4.5GB of memory bottles down to the
>>> question: what to do when the container starts to use more than assigned
>>> 4.5GB of memory?
>>>
>>> At this moment there are only 3 viable alternatives.
>>>
>>> A) Have separate memory management for each container,
>>> with separate buddy allocator, lru lists, page replacement mechanism.
>>> That implies a considerable overhead, and the main challenge there
>>> is sharing of pages between these separate memory managers.
>>>
>>> B) Return errors on extension of mappings, but not on page faults, where
>>> memory is actually consumed.
>>> In this case it makes sense to take into account not only the size
>>> of used
>>> memory, but the size of created mappings as well.
>>> This is approximately what "privvmpages" accounting/limiting
>>> provides in
>>> UBC.
>>>
>>> C) Rely on OOM killer.
>>> This is a fall-back method in UBC, for the case "privvmpages" limits
>>> still leave the possibility to overload the system.

>>>
>>
>>
>> D) Virtual scan of mm's in the over-limit container
>>
>> E) Modify existing physical scanner to be able to skip pages which
>>   belong to not-over-limit containers.
>>
>> F) Something else ;)
> We fully agree that other possible algorithms can and should exist.
> My idea only is that any of them would need accounting anyway
> (which is the most part of beancounters).
> Throtling, modified scanners etc. can be implemented as a separate
> BC parameters. Thus, an administrator will be able to select
> which policy should be applied to the container which is near its limit.
>
> So the patches I'm trying to send are a step-by-step accounting of all
> the resources and their simple limitations. More comprehensive limitation
> policy will be built on top of it later.
>

One of the issues I see is that bean counters are not very flexible. Tasks
cannot change bean counters dynamically after fork()/exec() that is - can they?

> BTW, UBC page beancounters allow to distinguish pages used by only one
> container and pages which are shared. So scanner can try to reclaim
> container private pages first, thus not influencing other containers.
>

But can you select the specific container for which we intend to scan pages?

> Thanks,
> Kirill
>

--
Thanks,
Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: Re: BC: resource beancounters (v2)
Posted by Rohit Seth on Tue, 29 Aug 2006 17:30:35 GMT
View Forum Message <> Reply to Message

On Tue, 2006-08-29 at 11:15 +0100, Alan Cox wrote:

> Ar Llu, 2006-08-28 am 15:28 -0700, ysgrifennodd Rohit Seth:
> > Though if we have file/directory based accounting then shared pages
> > belonging to /usr/lib or /usr/bin can go to a common container.
>
> So that one user can map all the spare libraries and config files and
> DoS the system by preventing people from accessing the libraries they do
> need ?
>

Well, there is a risk whenever there is sharing across containers. The
point though is, give the choice to sysadmin to configure the platform
the way it is appropriate.

-rohit

Subject: Re: Re: BC: resource beancounters (v2)
Posted by Alan Cox on Tue, 29 Aug 2006 18:46:07 GMT
View Forum Message <> Reply to Message

Ar Maw, 2006-08-29 am 10:30 -0700, ysgrifennodd Rohit Seth:
> On Tue, 2006-08-29 at 11:15 +0100, Alan Cox wrote:
> > Ar Llu, 2006-08-28 am 15:28 -0700, ysgrifennodd Rohit Seth:
> > > Though if we have file/directory based accounting then shared pages
> > > belonging to /usr/lib or /usr/bin can go to a common container.
> >
> > So that one user can map all the spare libraries and config files and
> > DoS the system by preventing people from accessing the libraries they do
> > need ?
> >
>
> Well, there is a risk whenever there is sharing across containers. The
> point though is, give the choice to sysadmin to configure the platform
> the way it is appropriate.

In other words your suggestion doesn't actually work for the real world
cases like web serving.

Subject: Re: Re: BC: resource beancounters (v2)
Posted by Rohit Seth on Tue, 29 Aug 2006 19:15:16 GMT
View Forum Message <> Reply to Message

On Tue, 2006-08-29 at 20:06 +0100, Alan Cox wrote:
> Ar Maw, 2006-08-29 am 10:30 -0700, ysgrifennodd Rohit Seth:
> > On Tue, 2006-08-29 at 11:15 +0100, Alan Cox wrote:
> > > Ar Llu, 2006-08-28 am 15:28 -0700, ysgrifennodd Rohit Seth:

> > > > Though if we have file/directory based accounting then shared pages
> > > > belonging to /usr/lib or /usr/bin can go to a common container.
> > >
> > > So that one user can map all the spare libraries and config files and
> > > DoS the system by preventing people from accessing the libraries they do
> > > need ?
> > >
> >
> > Well, there is a risk whenever there is sharing across containers. The
> > point though is, give the choice to sysadmin to configure the platform
> > the way it is appropriate.
>
> In other words your suggestion doesn't actually work for the real world
> cases like web serving.
>

Containers are not going to solve all the problems particularly the scenarios like when a machine is a web server and an odd user can log on to the same machine and (w/o any ulimits) claim all the memory that is present in the system.

Though it is quite possible to implement a combination of two (task and fs based) policies in containers and sysadmin can set a preference of each each container. [this probably is another reason for having a per page container pointer].

-rohit
