Subject: [RFC][PATCH] UBC: user resource beancounters
Posted by dev on Wed, 16 Aug 2006 15:23:58 GMT
View Forum Message <> Reply to Message

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

The full UBC 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. UBC allows 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 UBC and
management of kernel memory only. Virtual memory
management will be sent in a couple of days.

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

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

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

Task contains three beancounters:
1. exec_ub - current context. all resources are charged
to this beancounter.
2. task_ub - beancounter to which task_struct is charged
itself.
3. fork_sub - beancounter which is inherited by
task's children on fork

diff-ubc-syscalls.patch:
Patch adds system calls for UB management:
1. sys_getluid - get current UB id
2. sys_setluid - changes exec_ and fork_ UBs on current
3. sys_setublimit - set limits for resources consumtions

diff-ubc-kmem-core.patch:
Introduces UB_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_UBC flag - page
is charged to current's exec_ub.
2. Slabs - kmem_cache may be created with SLAB_UBC flag - in this
case each allocation is charged. Caches used by kmalloc are
created with SLAB_UBC | SLAB_UBC_NOCHARGE flags. In this case
only __GFP_UBC allocations are charged.

diff-ubc-kmem-charge.patch:
Adds SLAB_UBC and __GFP_UBC flags in appropriate places
to cause charging/limiting of specified resources.

diff-ubc-proc.patch:
Adds two proc entries user_beancounters and user_beancounters_sub
allowing to see current state (usage/limits/fails for each UB).
Implemented via seq files.

Patch set is applicable to 2.6.18-rc4-mm1

Thanks,
Kirill

Subject: [RFC][PATCH 1/7] UBC: kconfig
Posted by dev on Wed, 16 Aug 2006 15:34:19 GMT
View Forum Message <> Reply to Message

Add kernel/ub/Kconfig file with UBC options and
includes 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/ub/Kconfig | 25 +++++++++++++++++++++++++
 8 files changed, 39 insertions(+)

--- ./arch/i386/Kconfig.ubkm 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/ub/Kconfig"
+
 #
 # Use the generic interrupt handling code in kernel/irq/:
 #
--- ./arch/ia64/Kconfig.ubkm 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/ub/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 "ub/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 "ub/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 "ub/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 "lib/Kconfig"
--- ./arch/x86_64/Kconfig.ubkm 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/ub/Kconfig"
--- ./kernel/ub/Kconfig.ubkm 2006-07-28 13:07:38.000000000 +0400
+++ ./kernel/ub/Kconfig 2006-07-28 13:09:51.000000000 +0400
@@ -0,0 +1,25 @@
+#
+# User resources part (UBC)
+#
+# Copyright (C) 2006 OpenVZ. SWsoft Inc
+
```

```
+menu "User resources"
+
+config USER_RESOURCE
+ bool "Enable user resource accounting"
+ default y
+ 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: [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by dev on Wed, 16 Aug 2006 15:35:30 GMT
View Forum Message <> Reply to Message

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

Basic structures:
  ubparm   - resource description
  user_beancounter - set of resources, id, lock

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

---
 include/ub/beancounter.h | 157 ++++++++++++++++++
 init/main.c | 4
 kernel/Makefile | 1
 kernel/ub/Makefile | 7
 kernel/ub/beancounter.c | 398 +++
 5 files changed, 567 insertions(+)

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/ub/beancounter.h 2006-08-10 14:58:27.000000000 +0400
@@ -0,0 +1,157 @@

```
+/*
+ *  include/ub/beancounter.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#ifndef _LINUX_BEANCOUNTER_H
+#define _LINUX_BEANCOUNTER_H
+
+/*
+ * Resource list.
+ */
+
+#define UB_RESOURCES 0
+
+struct ubparm {
+ /*
+  * 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 barrier;
+ /* hard resource limit */
+ unsigned long limit;
+ /* consumed resources */
+ unsigned long held;
+ /* maximum amount of consumed resources through the last period */
+ unsigned long maxheld;
+ /* minimum amount of consumed resources through the last period */
+ unsigned long minheld;
+ /* count of failed charges */
+ unsigned long failcnt;
+};
+
+/*
+ * Kernel internal part.
+ */
+
+#ifdef __KERNEL__
+
+#include <linux/config.h>
+#include <linux/spinlock.h>
+#include <linux/list.h>
+#include <asm/atomic.h>
+
+/*
+ * UB_MAXVALUE is essentially LONG_MAX declared in a cross-compiling safe form.
```

```
+ */
+#define UB_MAXVALUE ( (1UL << (sizeof(unsigned long)*8-1)) - 1)
+
+
+/*
+ * Resource management structures
+ * Serialization issues:
+ *   beancounter list management is protected via ub_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-ub spinlock
+ */
+
+struct user_beancounter
+{
+ atomic_t  ub_refcount;
+ spinlock_t  ub_lock;
+ uid_t   ub_uid;
+ struct hlist_node hash;
+
+ struct user_beancounter *parent;
+ void   *private_data;
+
+ /* resources statistics and settings */
+ struct ubparm  ub_parms[UB_RESOURCES];
+};
+
+enum severity { UB_BARRIER, UB_LIMIT, UB_FORCE };
+
+/* Flags passed to beancounter_findcreate() */
+#define UB_LOOKUP_SUB  0x01 /* Lookup subbeancounter */
+#define UB_ALLOC  0x02 /* May allocate new one */
+#define UB_ALLOC_ATOMIC  0x04 /* Allocate with GFP_ATOMIC */
+
+#define UB_HASH_SIZE  256
+
+#ifdef CONFIG_USER_RESOURCE
+extern struct hlist_head ub_hash[];
+extern spinlock_t ub_hash_lock;
+
+static inline void ub_adjust_held_minmax(struct user_beancounter *ub,
+  int resource)
+{
+ if (ub->ub_parms[resource].maxheld < ub->ub_parms[resource].held)
+  ub->ub_parms[resource].maxheld = ub->ub_parms[resource].held;
+ if (ub->ub_parms[resource].minheld > ub->ub_parms[resource].held)
+  ub->ub_parms[resource].minheld = ub->ub_parms[resource].held;
+}
```

```
+
+void ub_print_resource_warning(struct user_beancounter *ub, int res,
+  char *str, unsigned long val, unsigned long held);
+void ub_print_uid(struct user_beancounter *ub, char *str, int size);
+
+int  __charge_beancounter_locked(struct user_beancounter *ub,
+  int resource, unsigned long val, enum severity strict);
+void charge_beancounter_notop(struct user_beancounter *ub,
+  int resource, unsigned long val);
+int  charge_beancounter(struct user_beancounter *ub,
+  int resource, unsigned long val, enum severity strict);
+
+void __uncharge_beancounter_locked(struct user_beancounter *ub,
+  int resource, unsigned long val);
+void uncharge_beancounter_notop(struct user_beancounter *ub,
+  int resource, unsigned long val);
+void uncharge_beancounter(struct user_beancounter *ub,
+  int resource, unsigned long val);
+
+struct user_beancounter *beancounter_findcreate(uid_t uid,
+  struct user_beancounter *parent, int flags);
+
+static inline struct user_beancounter *get_beancounter(
+  struct user_beancounter *ub)
+{
+ atomic_inc(&ub->ub_refcount);
+ return ub;
+}
+
+void __put_beancounter(struct user_beancounter *ub);
+static inline void put_beancounter(struct user_beancounter *ub)
+{
+ __put_beancounter(ub);
+}
+
+void ub_init_early(void);
+void ub_init_late(void);
+void ub_init_proc(void);
+
+extern struct user_beancounter ub0;
+extern const char *ub_rnames[];
+
+#else /* CONFIG_USER_RESOURCE */
+
+#define beancounter_findcreate(id, p, f)  (NULL)
+#define get_beancounter(ub)    (NULL)
+#define put_beancounter(ub)    do { } while (0)
+#define __charge_beancounter_locked(ub, r, v, s) (0)
```

```
+#define charge_beancounter(ub, r, v, s)   (0)
+#define charge_beancounter_notop(ub, r, v)  do { } while (0)
+#define __uncharge_beancounter_locked(ub, r, v)  do { } while (0)
+#define uncharge_beancounter(ub, r, v)   do { } while (0)
+#define uncharge_beancounter_notop(ub, r, v)  do { } while (0)
+#define ub_init_early()     do { } while (0)
+#define ub_init_late()     do { } while (0)
+#define ub_init_proc()     do { } while (0)
+
+#endif /* CONFIG_USER_RESOURCE */
+#endif /* __KERNEL__ */
+
+#endif /* _LINUX_BEANCOUNTER_H */
--- ./init/main.c.ubcore 2006-08-10 14:55:47.000000000 +0400
+++ ./init/main.c 2006-08-10 14:57:01.000000000 +0400
@@ -52,6 +52,8 @@
 #include <linux/debug_locks.h>
 #include <linux/lockdep.h>

+#include <ub/beancounter.h>
+
 #include <asm/io.h>
 #include <asm/bugs.h>
 #include <asm/setup.h>
@@ -470,6 +472,7 @@ asmlinkage void __init start_kernel(void
  early_boot_irqs_off();
  early_init_irq_lock_class();

+ ub_init_early();
 /*
  * Interrupts are still disabled. Do necessary setups, then
  * enable them
@@ -563,6 +566,7 @@ asmlinkage void __init start_kernel(void
 #endif
  fork_init(num_physpages);
  proc_caches_init();
+ ub_init_late();
  buffer_init();
  unnamed_dev_init();
  key_init();
--- ./kernel/Makefile.ubcore 2006-08-10 14:55:47.000000000 +0400
+++ ./kernel/Makefile 2006-08-10 14:57:01.000000000 +0400
@@ -12,6 +12,7 @@ obj-y     = sched.o fork.o exec_domain.o

 obj-$(CONFIG_STACKTRACE) += stacktrace.o
 obj-y += time/
+obj-y += ub/
 obj-$(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/ub/Makefile 2006-08-10 14:57:01.000000000 +0400
@@ -0,0 +1,7 @@
+#
+# User resources part (UBC)
+#
+# Copyright (C) 2006 OpenVZ. SWsoft Inc
+#
+
+obj-$(CONFIG_USER_RESOURCE) += beancounter.o
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/ub/beancounter.c 2006-08-10 15:09:34.000000000 +0400
@@ -0,0 +1,398 @@
+/*
+ *  kernel/ub/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 <ub/beancounter.h>
+
+static kmem_cache_t *ub_cachep;
+static struct user_beancounter default_beancounter;
+static struct user_beancounter default_subbeancounter;
+
+static void init_beancounter_struct(struct user_beancounter *ub, uid_t id);
+
+struct user_beancounter ub0;
+
+const char *ub_rnames[] = {
+};
+
+#define ub_hash_fun(x) ((((x) >> 8) ^ (x)) & (UB_HASH_SIZE - 1))
+#define ub_subhash_fun(p, id) ub_hash_fun((p)->ub_uid + (id) * 17)
+
+struct hlist_head ub_hash[UB_HASH_SIZE];
+spinlock_t ub_hash_lock;
+
+EXPORT_SYMBOL(ub_hash);
+EXPORT_SYMBOL(ub_hash_lock);
+
```

```
+/*
+ * Per user 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 user_beancounter *beancounter_findcreate(uid_t uid,
+  struct user_beancounter *p, int mask)
+{
+ struct user_beancounter *new_ub, *ub, *tmpl_ub;
+ unsigned long flags;
+ struct hlist_head *slot;
+ struct hlist_node *pos;
+
+ if (mask & UB_LOOKUP_SUB) {
+  WARN_ON(p == NULL);
+  tmpl_ub = &default_subbeancounter;
+  slot = &ub_hash[ub_subhash_fun(p, uid)];
+ } else {
+  WARN_ON(p != NULL);
+  tmpl_ub = &default_beancounter;
+  slot = &ub_hash[ub_hash_fun(uid)];
+ }
+ new_ub = NULL;
+
+retry:
+ spin_lock_irqsave(&ub_hash_lock, flags);
+ hlist_for_each_entry (ub, pos, slot, hash)
+  if (ub->ub_uid == uid && ub->parent == p)
+   break;
+
+ if (pos != NULL) {
+  get_beancounter(ub);
+  spin_unlock_irqrestore(&ub_hash_lock, flags);
+
+  if (new_ub != NULL) {
+   put_beancounter(new_ub->parent);
+   kmem_cache_free(ub_cachep, new_ub);
+  }
+  return ub;
+ }
+
```

```
+ if (!(mask & UB_ALLOC))
+  goto out_unlock;
+
+ if (new_ub != NULL)
+  goto out_install;
+
+ if (mask & UB_ALLOC_ATOMIC) {
+  new_ub = kmem_cache_alloc(ub_cachep, GFP_ATOMIC);
+  if (new_ub == NULL)
+   goto out_unlock;
+
+  memcpy(new_ub, tmpl_ub, sizeof(*new_ub));
+  init_beancounter_struct(new_ub, uid);
+  if (p)
+   new_ub->parent = get_beancounter(p);
+  goto out_install;
+ }
+
+ spin_unlock_irqrestore(&ub_hash_lock, flags);
+
+ new_ub = kmem_cache_alloc(ub_cachep, GFP_KERNEL);
+ if (new_ub == NULL)
+  goto out;
+
+ memcpy(new_ub, tmpl_ub, sizeof(*new_ub));
+ init_beancounter_struct(new_ub, uid);
+ if (p)
+  new_ub->parent = get_beancounter(p);
+ goto retry;
+
+out_install:
+ hlist_add_head(&new_ub->hash, slot);
+out_unlock:
+ spin_unlock_irqrestore(&ub_hash_lock, flags);
+out:
+ return new_ub;
+}
+
+EXPORT_SYMBOL(beancounter_findcreate);
+
+void ub_print_uid(struct user_beancounter *ub, char *str, int size)
+{
+ if (ub->parent != NULL)
+  snprintf(str, size, "%u.%u", ub->parent->ub_uid, ub->ub_uid);
+ else
+  snprintf(str, size, "%u", ub->ub_uid);
+}
+
```

```
+EXPORT_SYMBOL(ub_print_uid);
+
+void ub_print_resource_warning(struct user_beancounter *ub, int res,
+  char *str, unsigned long val, unsigned long held)
+{
+ char uid[64];
+
+ ub_print_uid(ub, uid, sizeof(uid));
+ printk(KERN_WARNING "UB %s %s warning: %s "
+   "(held %lu, fails %lu, val %lu)\n",
+   uid, ub_rnames[res], str,
+   (res < UB_RESOURCES ? ub->ub_parms[res].held : held),
+   (res < UB_RESOURCES ? ub->ub_parms[res].failcnt : 0),
+   val);
+}
+
+EXPORT_SYMBOL(ub_print_resource_warning);
+
+static inline void verify_held(struct user_beancounter *ub)
+{
+ int i;
+
+ for (i = 0; i < UB_RESOURCES; i++)
+  if (ub->ub_parms[i].held != 0)
+   ub_print_resource_warning(ub, i,
+     "resource is held on put", 0, 0);
+}
+
+void __put_beancounter(struct user_beancounter *ub)
+{
+ unsigned long flags;
+ struct user_beancounter *parent;
+
+again:
+ parent = ub->parent;
+ /* equevalent to atomic_dec_and_lock_irqsave() */
+ local_irq_save(flags);
+ if (likely(!atomic_dec_and_lock(&ub->ub_refcount, &ub_hash_lock))) {
+  if (unlikely(atomic_read(&ub->ub_refcount) < 0))
+   printk(KERN_ERR "UB: Bad ub refcount: ub=%p, "
+     "luid=%d, ref=%d\n",
+     ub, ub->ub_uid,
+     atomic_read(&ub->ub_refcount));
+  local_irq_restore(flags);
+  return;
+ }
+
+ if (unlikely(ub == &ub0)) {
```

```
+  printk(KERN_ERR "Trying to put ub0\n");
+  spin_unlock_irqrestore(&ub_hash_lock, flags);
+  return;
+ }
+
+ verify_held(ub);
+ hlist_del(&ub->hash);
+ spin_unlock_irqrestore(&ub_hash_lock, flags);
+
+ kmem_cache_free(ub_cachep, ub);
+
+ ub = parent;
+ if (ub != NULL)
+  goto again;
+}
+
+EXPORT_SYMBOL(__put_beancounter);
+
+/*
+ * Generic resource charging stuff
+ */
+
+int __charge_beancounter_locked(struct user_beancounter *ub,
+  int resource, unsigned long val, enum severity strict)
+{
+ /*
+  * ub_value <= UB_MAXVALUE, value <= UB_MAXVALUE, and only one addition
+  * at the moment is possible so an overflow is impossible.
+  */
+ ub->ub_parms[resource].held += val;
+
+ switch (strict) {
+  case UB_BARRIER:
+   if (ub->ub_parms[resource].held >
+     ub->ub_parms[resource].barrier)
+    break;
+   /* fallthrough */
+  case UB_LIMIT:
+   if (ub->ub_parms[resource].held >
+     ub->ub_parms[resource].limit)
+    break;
+   /* fallthrough */
+  case UB_FORCE:
+   ub_adjust_held_minmax(ub, resource);
+   return 0;
+  default:
+   BUG();
+ }
```

```
+
+ ub->ub_parms[resource].failcnt++;
+ ub->ub_parms[resource].held -= val;
+ return -ENOMEM;
+}
+
+int charge_beancounter(struct user_beancounter *ub,
+  int resource, unsigned long val, enum severity strict)
+{
+ int retval;
+ struct user_beancounter *p, *q;
+ unsigned long flags;
+
+ retval = -EINVAL;
+ BUG_ON(val > UB_MAXVALUE);
+
+ local_irq_save(flags);
+ for (p = ub; p != NULL; p = p->parent) {
+  spin_lock(&p->ub_lock);
+  retval = __charge_beancounter_locked(p, resource, val, strict);
+  spin_unlock(&p->ub_lock);
+  if (retval)
+   goto unroll;
+ }
+out_restore:
+ local_irq_restore(flags);
+ return retval;
+
+unroll:
+ for (q = ub; q != p; q = q->parent) {
+  spin_lock(&q->ub_lock);
+  __uncharge_beancounter_locked(q, resource, val);
+  spin_unlock(&q->ub_lock);
+ }
+ goto out_restore;
+}
+
+EXPORT_SYMBOL(charge_beancounter);
+
+void charge_beancounter_notop(struct user_beancounter *ub,
+  int resource, unsigned long val)
+{
+ struct user_beancounter *p;
+ unsigned long flags;
+
+ local_irq_save(flags);
+ for (p = ub; p->parent != NULL; p = p->parent) {
+  spin_lock(&p->ub_lock);
```

```
+  __charge_beancounter_locked(p, resource, val, UB_FORCE);
+  spin_unlock(&p->ub_lock);
+ }
+ local_irq_restore(flags);
+}
+
+EXPORT_SYMBOL(charge_beancounter_notop);
+
+void __uncharge_beancounter_locked(struct user_beancounter *ub,
+  int resource, unsigned long val)
+{
+ if (unlikely(ub->ub_parms[resource].held < val)) {
+  ub_print_resource_warning(ub, resource,
+    "uncharging too much", val, 0);
+  val = ub->ub_parms[resource].held;
+ }
+ ub->ub_parms[resource].held -= val;
+ ub_adjust_held_minmax(ub, resource);
+}
+
+void uncharge_beancounter(struct user_beancounter *ub,
+  int resource, unsigned long val)
+{
+ unsigned long flags;
+ struct user_beancounter *p;
+
+ for (p = ub; p != NULL; p = p->parent) {
+  spin_lock_irqsave(&p->ub_lock, flags);
+  __uncharge_beancounter_locked(p, resource, val);
+  spin_unlock_irqrestore(&p->ub_lock, flags);
+ }
+}
+
+EXPORT_SYMBOL(uncharge_beancounter);
+
+void uncharge_beancounter_notop(struct user_beancounter *ub,
+  int resource, unsigned long val)
+{
+ struct user_beancounter *p;
+ unsigned long flags;
+
+ local_irq_save(flags);
+ for (p = ub; p->parent != NULL; p = p->parent) {
+  spin_lock(&p->ub_lock);
+  __uncharge_beancounter_locked(p, resource, val);
+  spin_unlock(&p->ub_lock);
+ }
+ local_irq_restore(flags);
```

```
+}
+
+EXPORT_SYMBOL(uncharge_beancounter_notop);
+
+/*
+ * Initialization
+ *
+ * struct user_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 user_beancounter *ub, uid_t id)
+{
+ atomic_set(&ub->ub_refcount, 1);
+ spin_lock_init(&ub->ub_lock);
+ ub->ub_uid = id;
+}
+
+static void init_beancounter_nolimits(struct user_beancounter *ub)
+{
+ int k;
+
+ for (k = 0; k < UB_RESOURCES; k++) {
+  ub->ub_parms[k].limit = UB_MAXVALUE;
+  ub->ub_parms[k].barrier = UB_MAXVALUE;
+ }
+}
+
+static void init_beancounter_syslimits(struct user_beancounter *ub)
+{
+ int k;
+
+ for (k = 0; k < UB_RESOURCES; k++)
+  ub->ub_parms[k].barrier = ub->ub_parms[k].limit;
+}
+
+void __init ub_init_early(void)
+{
+ struct user_beancounter *ub;
+ struct hlist_head *slot;
+
+ ub = &ub0;
+
```

```
+ memset(ub, 0, sizeof(*ub));
+ init_beancounter_nolimits(ub);
+ init_beancounter_struct(ub, 0);
+
+ spin_lock_init(&ub_hash_lock);
+ slot = &ub_hash[ub_hash_fun(ub->ub_uid)];
+ hlist_add_head(&ub->hash, slot);
+}
+
+void __init ub_init_late(void)
+{
+ struct user_beancounter *ub;
+
+ ub_cachep = kmem_cache_create("user_beancounters",
+   sizeof(struct user_beancounter),
+   0, SLAB_HWCACHE_ALIGN, NULL, NULL);
+ if (ub_cachep == NULL)
+  panic("Can't create ubc caches\n");
+
+ ub = &default_beancounter;
+ memset(ub, 0, sizeof(default_beancounter));
+ init_beancounter_syslimits(ub);
+ init_beancounter_struct(ub, 0);
+
+ ub = &default_subbeancounter;
+ memset(ub, 0, sizeof(default_subbeancounter));
+ init_beancounter_nolimits(ub);
+ init_beancounter_struct(ub, 0);
+}
```

Subject: [RFC][PATCH 3/7] UBC: ub context and inheritance
Posted by dev on Wed, 16 Aug 2006 15:36:42 GMT
View Forum Message <> Reply to Message

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

Task references three beancounters:
1. exec_ub current context. all resources are
   charged to this beancounter.
2. task_ub beancounter to which task_struct is
   charged itself.
3. fork_sub 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/ub/task.h       |   42 ++++++++++++++++++++++++++++++++++++++++++
 kernel/fork.c           |   21 +++++++++++++++++++-----
 kernel/irq/handle.c     |    9 +++++++++
 kernel/softirq.c        |    8 ++++++++
 kernel/ub/Makefile      |    1 +
 kernel/ub/beancounter.c |    4 ++++
 kernel/ub/misc.c        |   34 ++++++++++++++++++++++++++++++++++
 8 files changed, 119 insertions(+), 5 deletions(-)

--- ./include/linux/sched.h.ubfork 2006-07-17 17:01:12.000000000 +0400
+++ ./include/linux/sched.h 2006-07-31 16:01:54.000000000 +0400
@@ -81,6 +81,8 @@ struct sched_param {
 #include <linux/timer.h>
 #include <linux/hrtimer.h>

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

 struct exec_domain;
@@ -997,6 +999,9 @@ struct task_struct {
  spinlock_t delays_lock;
  struct task_delay_info *delays;
 #endif
+#ifdef CONFIG_USER_RESOURCE
+ struct task_beancounter task_bc;
+#endif
 };

 static inline pid_t process_group(struct task_struct *tsk)
--- ./include/ub/task.h.ubfork 2006-07-28 18:53:52.000000000 +0400
+++ ./include/ub/task.h 2006-08-01 15:26:08.000000000 +0400
@@ -0,0 +1,42 @@
+/*
+ *  include/ub/task.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#ifndef __UB_TASK_H_
+#define __UB_TASK_H_
+
+#include <linux/config.h>
+
```

```
+struct user_beancounter;
+
+struct task_beancounter {
+ struct user_beancounter *exec_ub;
+ struct user_beancounter *task_ub;
+ struct user_beancounter *fork_sub;
+};
+
+#ifdef CONFIG_USER_RESOURCE
+#define get_exec_ub()  (current->task_bc.exec_ub)
+#define set_exec_ub(newub)   \
+ ({     \
+   struct user_beancounter *old; \
+   struct task_beancounter *tbc; \
+   tbc = &current->task_bc; \
+   old = tbc->exec_ub;  \
+   tbc->exec_ub = newub;  \
+   old;    \
+  })
+
+int ub_task_charge(struct task_struct *parent, struct task_struct *new);
+void ub_task_uncharge(struct task_struct *tsk);
+
+#else /* CONFIG_USER_RESOURCE */
+#define get_exec_ub()  (NULL)
+#define set_exec_ub(__ub) (NULL)
+#define ub_task_charge(p, t) (0)
+#define ub_task_uncharge(t) do { } while (0)
+#endif /* CONFIG_USER_RESOURCE */
+#endif /* __UB_TASK_H_ */
--- ./kernel/irq/handle.c.ubirq 2006-07-10 12:39:20.000000000 +0400
+++ ./kernel/irq/handle.c 2006-08-01 12:39:34.000000000 +0400
@@ -16,6 +16,9 @@
 #include <linux/interrupt.h>
 #include <linux/kernel_stat.h>

+#include <ub/beancounter.h>
+#include <ub/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 user_beancounter *ub;
+
```

```
+ ub = set_exec_ub(&ub0);

  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);
+
+  (void) set_exec_ub(ub);
   return 1;
  }

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

+ (void) set_exec_ub(ub);
  return 1;
 }

--- ./kernel/softirq.c.ubirq 2006-07-17 17:01:12.000000000 +0400
+++ ./kernel/softirq.c 2006-08-01 12:40:44.000000000 +0400
@@ -18,6 +18,9 @@
 #include <linux/rcupdate.h>
 #include <linux/smp.h>

+#include <ub/beancounter.h>
+#include <ub/task.h>
+
 #include <asm/irq.h>
 /*
    - No shared variables, all the data are CPU local.
@@ -191,6 +194,9 @@ asmlinkage void __do_softirq(void)
  __u32 pending;
  int max_restart = MAX_SOFTIRQ_RESTART;
  int cpu;
+ struct user_beancounter *ub;
+
+ ub = set_exec_ub(&ub0);

  pending = local_softirq_pending();
  account_system_vtime(current);
@@ -229,6 +235,8 @@ restart:

  account_system_vtime(current);
  _local_bh_enable();
+
```

```
+ (void) set_exec_ub(ub);
 }

 #ifndef __ARCH_HAS_DO_SOFTIRQ
--- ./kernel/fork.c.ubfork 2006-07-17 17:01:12.000000000 +0400
+++ ./kernel/fork.c 2006-08-01 12:58:36.000000000 +0400
@@ -46,6 +46,8 @@
 #include <linux/delayacct.h>
 #include <linux/taskstats_kern.h>

+#include <ub/task.h>
+
 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
 #include <asm/uaccess.h>
@@ -102,6 +104,7 @@ static kmem_cache_t *mm_cachep;

 void free_task(struct task_struct *tsk)
 {
+ ub_task_uncharge(tsk);
  free_thread_info(tsk->thread_info);
  rt_mutex_debug_task_free(tsk);
  free_task_struct(tsk);
@@ -162,18 +165,19 @@ static struct task_struct *dup_task_stru

  tsk = alloc_task_struct();
  if (!tsk)
-  return NULL;
+  goto out;

  ti = alloc_thread_info(tsk);
- if (!ti) {
-  free_task_struct(tsk);
-  return NULL;
- }
+ if (!ti)
+  goto out_tsk;

  *tsk = *orig;
  tsk->thread_info = ti;
  setup_thread_stack(tsk, orig);

+ if (ub_task_charge(orig, tsk))
+  goto out_ti;
+
  /* One for us, one for whoever does the "release_task()" (usually parent) */
  atomic_set(&tsk->usage,2);
  atomic_set(&tsk->fs_excl, 0);
```

```
@@ -180,6 +184,13 @@ static struct task_struct *dup_task_stru
 #endif
  tsk->splice_pipe = NULL;
  return tsk;
+
+out_ti:
+ free_thread_info(ti);
+out_tsk:
+ free_task_struct(tsk);
+out:
+ return NULL;
 }

 #ifdef CONFIG_MMU
--- ./kernel/ub/Makefile.ubcore 2006-08-03 16:24:56.000000000 +0400
+++ ./kernel/ub/Makefile 2006-08-01 11:08:39.000000000 +0400
@@ -5,3 +5,4 @@
 #

 obj-$(CONFIG_USER_RESOURCE) += beancounter.o
+obj-$(CONFIG_USER_RESOURCE) += misc.o
--- ./kernel/ub/beancounter.c.ubcore 2006-07-28 13:07:44.000000000 +0400
+++ ./kernel/ub/beancounter.c 2006-08-03 16:14:17.000000000 +0400
@@ -395,6 +395,10 @@
  spin_lock_init(&ub_hash_lock);
  slot = &ub_hash[ub_hash_fun(ub->ub_uid)];
  hlist_add_head(&ub->hash, slot);
+
+ current->task_bc.exec_ub = ub;
+ current->task_bc.task_ub = get_beancounter(ub);
+ current->task_bc.fork_sub = get_beancounter(ub);
 }

 void __init ub_init_late(void)
--- ./kernel/ub/misc.c.ubfork 2006-07-31 16:23:44.000000000 +0400
+++ ./kernel/ub/misc.c 2006-07-31 16:28:47.000000000 +0400
@@ -0,0 +1,34 @@
+/*
+ * kernel/ub/misc.c
+ *
+ * Copyright (C) 2006 OpenVZ. SWsoft Inc.
+ *
+ */
+
+#include <linux/sched.h>
+
+#include <ub/beancounter.h>
+#include <ub/task.h>
```

```
+
+int ub_task_charge(struct task_struct *parent, struct task_struct *new)
+{
+ struct task_beancounter *old_bc;
+ struct task_beancounter *new_bc;
+ struct user_beancounter *ub;
+
+ old_bc = &parent->task_bc;
+ new_bc = &new->task_bc;
+
+ ub = old_bc->fork_sub;
+ new_bc->exec_ub = get_beancounter(ub);
+ new_bc->task_ub = get_beancounter(ub);
+ new_bc->fork_sub = get_beancounter(ub);
+ return 0;
+}
+
+void ub_task_uncharge(struct task_struct *tsk)
+{
+ put_beancounter(tsk->task_bc.exec_ub);
+ put_beancounter(tsk->task_bc.task_ub);
+ put_beancounter(tsk->task_bc.fork_sub);
+}
```

Subject: [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by dev on Wed, 16 Aug 2006 15:37:43 GMT
View Forum Message <> Reply to Message

Add the following system calls for UB management:
1. sys_getluid - get current UB id
2. sys_setluid - changes exec_ and fork_ UBs on current
3. sys_setublimit - set limits for resources consumtions

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

```
 arch/i386/kernel/syscall_table.S |    3
 arch/ia64/kernel/entry.S         |    3
 arch/sparc/kernel/systbls.S      |    2
 arch/sparc64/kernel/systbls.S    |    2
 include/asm-i386/unistd.h        |    5 +
 include/asm-ia64/unistd.h        |    5 +
 include/asm-powerpc/systbl.h     |    3
 include/asm-powerpc/unistd.h     |    5 +
 include/asm-sparc/unistd.h       |    3
 include/asm-sparc64/unistd.h     |    3
```

```
 include/asm-x86_64/unistd.h      |    8 ++
 kernel/ub/Makefile               |    1
 kernel/ub/sys.c                  |  126 ++++++++++++++++++++++++++++++++++++++++
 13 files changed, 163 insertions(+), 6 deletions(-)

--- ./arch/i386/kernel/syscall_table.S.ubsys 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/i386/kernel/syscall_table.S 2006-07-31 14:36:59.000000000 +0400
@@ -317,3 +317,6 @@ ENTRY(sys_call_table)
  .long sys_vmsplice
  .long sys_move_pages
  .long sys_getcpu
+ .long sys_getluid
+ .long sys_setluid
+ .long sys_setublimit  /* 320 */
--- ./arch/ia64/kernel/entry.S.ubsys 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/ia64/kernel/entry.S 2006-07-31 15:25:36.000000000 +0400
@@ -1610,5 +1610,8 @@ sys_call_table:
  data8 sys_sync_file_range  // 1300
  data8 sys_tee
  data8 sys_vmsplice
+ daat8 sys_getluid
+ data8 sys_setluid
+ data8 sys_setublimit   // 1305

  .org sys_call_table + 8*NR_syscalls // guard against failures to increase NR_syscalls
--- ./arch/sparc/kernel/systbls.S.arsys 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/sparc/kernel/systbls.S 2006-08-10 17:07:15.000000000 +0400
@@ -78,7 +78,7 @@ 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_getluid, sys_setluid, sys_setublimit

 #ifdef CONFIG_SUNOS_EMUL
  /* Now the SunOS syscall table. */
--- ./arch/sparc64/kernel/systbls.S.arsys 2006-07-10 12:39:11.000000000 +0400
+++ ./arch/sparc64/kernel/systbls.S 2006-08-10 17:08:52.000000000 +0400
@@ -79,7 +79,7 @@ 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_getluid, sys_setluid,
sys_setublimit

 #endif /* CONFIG_COMPAT */
```

```
--- ./include/asm-i386/unistd.h.ubsys 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-i386/unistd.h 2006-07-31 15:56:31.000000000 +0400
@@ -323,10 +323,13 @@
 #define __NR_vmsplice  316
 #define __NR_move_pages  317
 #define __NR_getcpu  318
+#define __NR_getluid  319
+#define __NR_setluid  320
+#define __NR_setublimit  321

 #ifdef __KERNEL__

-#define NR_syscalls 318
+#define NR_syscalls 322
 #include <linux/err.h>

 /*
--- ./include/asm-ia64/unistd.h.ubsys 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-ia64/unistd.h 2006-07-31 15:57:23.000000000 +0400
@@ -291,11 +291,14 @@
 #define __NR_sync_file_range  1300
 #define __NR_tee   1301
 #define __NR_vmsplice   1302
+#define __NR_getluid   1303
+#define __NR_setluid   1304
+#define __NR_setublimit   1305

 #ifdef __KERNEL__


-#define NR_syscalls   279 /* length of syscall table */
+#define NR_syscalls   282 /* length of syscall table */

 #define __ARCH_WANT_SYS_RT_SIGACTION

--- ./include/asm-powerpc/systbl.h.arsys 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-powerpc/systbl.h 2006-08-10 17:05:53.000000000 +0400
@@ -304,3 +304,6 @@ SYSCALL_SPU(fchmodat)
 SYSCALL_SPU(faccessat)
 COMPAT_SYS_SPU(get_robust_list)
 COMPAT_SYS_SPU(set_robust_list)
+SYSCALL(sys_getluid)
+SYSCALL(sys_setluid)
+SYSCALL(sys_setublimit)
--- ./include/asm-powerpc/unistd.h.arsys 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-powerpc/unistd.h 2006-08-10 17:06:28.000000000 +0400
@@ -323,10 +323,13 @@
 #define __NR_faccessat  298
```

```
 #define __NR_get_robust_list 299
 #define __NR_set_robust_list 300
+#define __NR_getluid  301
+#define __NR_setluid  302
+#define __NR_setublimit  303

 #ifdef __KERNEL__

-#define __NR_syscalls  301
+#define __NR_syscalls  304

 #define __NR__exit __NR_exit
 #define NR_syscalls __NR_syscalls
--- ./include/asm-sparc/unistd.h.arsys 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-sparc/unistd.h 2006-08-10 17:08:19.000000000 +0400
@@ -318,6 +318,9 @@
 #define __NR_unshare  299
 #define __NR_set_robust_list 300
 #define __NR_get_robust_list 301
+#define __NR_getluid  302
+#define __NR_setluid  303
+#define __NR_setublimit  304

 #ifdef __KERNEL__
 /* WARNING: You MAY NOT add syscall numbers larger than 301, since
--- ./include/asm-sparc64/unistd.h.arsys 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-sparc64/unistd.h 2006-08-10 17:09:24.000000000 +0400
@@ -320,6 +320,9 @@
 #define __NR_unshare  299
 #define __NR_set_robust_list 300
 #define __NR_get_robust_list 301
+#define __NR_getluid  302
+#define __NR_setluid  303
+#define __NR_setublimit  304

 #ifdef __KERNEL__
 /* WARNING: You MAY NOT add syscall numbers larger than 301, since
--- ./include/asm-x86_64/unistd.h.ubsys 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-x86_64/unistd.h 2006-07-31 16:00:01.000000000 +0400
@@ -619,10 +619,16 @@ __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_getluid  280
+__SYSCALL(__NR_getluid, sys_getluid)
+#define __NR_setluid  281
+__SYSCALL(__NR_setluid, sys_setluid)
+#define __NR_setublimit  282
```

```
+__SYSCALL(__NR_setublimit, sys_setublimit)

 #ifdef __KERNEL__

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

 #ifndef __NO_STUBS
--- ./kernel/ub/Makefile.ubsys 2006-07-28 14:08:37.000000000 +0400
+++ ./kernel/ub/Makefile 2006-08-01 11:08:39.000000000 +0400
@@ -6,3 +6,4 @@

 obj-$(CONFIG_USER_RESOURCE) += beancounter.o
 obj-$(CONFIG_USER_RESOURCE) += misc.o
+obj-y += sys.o
--- ./kernel/ub/sys.c.ubsys 2006-07-28 18:52:18.000000000 +0400
+++ ./kernel/ub/sys.c 2006-08-03 16:14:23.000000000 +0400
@@ -0,0 +1,126 @@
+/*
+ *  kernel/ub/sys.c
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#include <linux/config.h>
+#include <linux/sched.h>
+#include <asm/uaccess.h>
+
+#include <ub/beancounter.h>
+#include <ub/task.h>
+
+#ifndef CONFIG_USER_RESOURCE
+asmlinkage long sys_getluid(void)
+{
+ return -ENOSYS;
+}
+
+asmlinkage long sys_setluid(uid_t uid)
+{
+ return -ENOSYS;
+}
+
+asmlinkage long sys_setublimit(uid_t uid, unsigned long resource,
+  unsigned long *limits)
+{
+ return -ENOSYS;
```

```
+}
+#else /* CONFIG_USER_RESOURCE */
+
+/*
+ * The (rather boring) getluid syscall
+ */
+asmlinkage long sys_getluid(void)
+{
+ struct user_beancounter *ub;
+
+ ub = get_exec_ub();
+ if (ub == NULL)
+  return -EINVAL;
+
+ return ub->ub_uid;
+}
+
+/*
+ * The setluid syscall
+ */
+asmlinkage long sys_setluid(uid_t uid)
+{
+ int error;
+ struct user_beancounter *ub;
+ struct task_beancounter *task_bc;
+
+ task_bc = &current->task_bc;
+
+ /* You may not disown a setluid */
+ error = -EINVAL;
+ if (uid == (uid_t)-1)
+  goto out;
+
+ /* You may only set an ub as root */
+ error = -EPERM;
+ if (!capable(CAP_SETUID))
+  goto out;
+
+ /* Ok - set up a beancounter entry for this user */
+ error = -ENOBUFS;
+ ub = beancounter_findcreate(uid, NULL, UB_ALLOC);
+ if (ub == NULL)
+  goto out;
+
+ /* install bc */
+ put_beancounter(task_bc->exec_ub);
+ task_bc->exec_ub = ub;
+ put_beancounter(task_bc->fork_sub);
```

```
+ task_bc->fork_sub = get_beancounter(ub);
+ error = 0;
+out:
+ return error;
+}
+
+/*
+ * The setbeanlimit syscall
+ */
+asmlinkage long sys_setublimit(uid_t uid, unsigned long resource,
+  unsigned long *limits)
+{
+ int error;
+ unsigned long flags;
+ struct user_beancounter *ub;
+ unsigned long new_limits[2];
+
+ error = -EPERM;
+ if(!capable(CAP_SYS_RESOURCE))
+  goto out;
+
+ error = -EINVAL;
+ if (resource >= UB_RESOURCES)
+  goto out;
+
+ error = -EFAULT;
+ if (copy_from_user(&new_limits, limits, sizeof(new_limits)))
+  goto out;
+
+ error = -EINVAL;
+ if (new_limits[0] > UB_MAXVALUE || new_limits[1] > UB_MAXVALUE)
+  goto out;
+
+ error = -ENOENT;
+ ub = beancounter_findcreate(uid, NULL, 0);
+ if (ub == NULL)
+  goto out;
+
+ spin_lock_irqsave(&ub->ub_lock, flags);
+ ub->ub_parms[resource].barrier = new_limits[0];
+ ub->ub_parms[resource].limit = new_limits[1];
+ spin_unlock_irqrestore(&ub->ub_lock, flags);
+
+ put_beancounter(ub);
+ error = 0;
+out:
+ return error;
+}
```

+#endif

Subject: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by dev on Wed, 16 Aug 2006 15:39:03 GMT
View Forum Message <> Reply to Message

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

Reference to UB 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_UBC flag - page
   is charged to current's exec_ub.
2. Slabs - kmem_cache may be created with SLAB_UBC flag - in this
   case each allocation is charged. Caches used by kmalloc are
   created with SLAB_UBC | SLAB_UBC_NOCHARGE flags. In this case
   only __GFP_UBC 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/ub/beancounter.h | 4 +
 include/ub/kmem.h | 33 ++++++++++++
 kernel/ub/Makefile | 1
 kernel/ub/beancounter.c | 3 +
 kernel/ub/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.kmemcore 2006-08-16 19:10:38.000000000 +0400
+++ ./include/linux/gfp.h 2006-08-16 19:12:56.000000000 +0400
@@ -46,15 +46,18 @@ struct vm_area_struct;
 #define __GFP_NOMEMALLOC ((__force gfp_t)0x10000u) /* Don't use emergency reserves */
 #define __GFP_HARDWALL   ((__force gfp_t)0x20000u) /* Enforce hardwall cpuset memory allocs */
 #define __GFP_THISNODE ((__force gfp_t)0x40000u)/* No fallback, no policies */

```
+#define __GFP_UBC ((__force gfp_t)0x80000u) /* Charge allocation with UB */
+#define __GFP_UBC_LIMIT ((__force gfp_t)0x100000u) /* Charge against UB limit */

-#define __GFP_BITS_SHIFT 20 /* Room for 20 __GFP_FOO bits */
+#define __GFP_BITS_SHIFT 21 /* Room for 20 __GFP_FOO bits */
 #define __GFP_BITS_MASK ((__force gfp_t)((1 << __GFP_BITS_SHIFT) - 1))

 /* if you forget to add the bitmask here kernel will crash, period */
 #define GFP_LEVEL_MASK (__GFP_WAIT|__GFP_HIGH|__GFP_IO|__GFP_FS| \
    __GFP_COLD|__GFP_NOWARN|__GFP_REPEAT| \
    __GFP_NOFAIL|__GFP_NORETRY|__GFP_NO_GROW|__GFP_COMP| \
-   __GFP_NOMEMALLOC|__GFP_HARDWALL|__GFP_THISNODE)
+   __GFP_NOMEMALLOC|__GFP_HARDWALL|__GFP_THISNODE| \
+   __GFP_UBC|__GFP_UBC_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_UBC (__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_UBC)
 #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.kmemcore 2006-08-16 19:10:38.000000000 +0400
+++ ./include/linux/mm.h 2006-08-16 19:10:51.000000000 +0400
@@ -274,8 +274,14 @@ struct page {
  unsigned int gfp_mask;
  unsigned long trace[8];
 #endif
+#ifdef CONFIG_USER_RESOURCE
+ union {
+  struct user_beancounter *page_ub;
+ } bc;
+#endif
 };

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

--- ./include/linux/slab.h.kmemcore 2006-08-16 19:10:38.000000000 +0400
+++ ./include/linux/slab.h 2006-08-16 19:10:51.000000000 +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_UBC  0x00200000UL /* Account with UB */
+#define SLAB_UBC_NOCHARGE 0x00400000UL /* Explicit accounting */

 /* flags passed to a constructor func */
 #define SLAB_CTOR_CONSTRUCTOR 0x001UL  /* if not set, then deconstructor */
@@ -293,6 +295,8 @@ extern kmem_cache_t *bio_cachep;

 extern atomic_t slab_reclaim_pages;

+struct user_beancounter;
+struct user_beancounter **kmem_cache_ubp(kmem_cache_t *cachep, void *obj);
 #endif /* __KERNEL__ */

 #endif /* _LINUX_SLAB_H */
--- ./include/linux/vmalloc.h.kmemcore 2006-08-16 19:10:38.000000000 +0400
+++ ./include/linux/vmalloc.h 2006-08-16 19:10:51.000000000 +0400
@@ -36,6 +36,7 @@ struct vm_struct {
  * Highlevel APIs for driver use
  */
 extern void *vmalloc(unsigned long size);
+extern void *vmalloc_ub(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/ub/beancounter.h.kmemcore 2006-08-16 19:10:38.000000000 +0400
+++ ./include/ub/beancounter.h 2006-08-16 19:10:51.000000000 +0400
@@ -12,7 +12,9 @@
  * Resource list.
  */

-#define UB_RESOURCES 0
+#define UB_KMEMSIZE 0
+
+#define UB_RESOURCES 1

 struct ubparm {
  /*
--- ./include/ub/kmem.h.kmemcore 2006-08-16 19:10:38.000000000 +0400
+++ ./include/ub/kmem.h 2006-08-16 19:10:51.000000000 +0400
@@ -0,0 +1,33 @@
+/*
+ *  include/ub/kmem.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#ifndef __UB_KMEM_H_
```

```
+#define __UB_KMEM_H_
+
+#include <linux/config.h>
+
+/*
+ * UB_KMEMSIZE accounting
+ */
+
+struct mm_struct;
+struct page;
+struct user_beancounter;
+
+#ifdef CONFIG_USER_RESOURCE
+int  ub_page_charge(struct page *page, int order, gfp_t flags);
+void ub_page_uncharge(struct page *page, int order);
+
+int ub_slab_charge(kmem_cache_t *cachep, void *obj, gfp_t flags);
+void ub_slab_uncharge(kmem_cache_t *cachep, void *obj);
+#else
+#define ub_page_charge(pg, o, mask) (0)
+#define ub_page_uncharge(pg, o)  do { } while (0)
+#define ub_slab_charge(cachep, o) (0)
+#define ub_slab_uncharge(cachep, o) do { } while (0)
+#endif
+#endif /* __UB_SLAB_H_ */
--- ./kernel/ub/Makefile.kmemcore 2006-08-16 19:10:38.000000000 +0400
+++ ./kernel/ub/Makefile 2006-08-16 19:10:51.000000000 +0400
@@ -7,3 +7,4 @@
 obj-$(CONFIG_USER_RESOURCE) += beancounter.o
 obj-$(CONFIG_USER_RESOURCE) += misc.o
 obj-y += sys.o
+obj-$(CONFIG_USER_RESOURCE) += kmem.o
--- ./kernel/ub/beancounter.c.kmemcore 2006-08-16 19:10:38.000000000 +0400
+++ ./kernel/ub/beancounter.c 2006-08-16 19:10:51.000000000 +0400
@@ -20,6 +20,7 @@ static void init_beancounter_struct(stru
 struct user_beancounter ub0;

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

 #define ub_hash_fun(x) ((((x) >> 8) ^ (x)) & (UB_HASH_SIZE - 1))
@@ -356,6 +357,8 @@ static void init_beancounter_syslimits(s
 {
  int k;

+ ub->ub_parms[UB_KMEMSIZE].limit = 32 * 1024 * 1024;
+
```

```
  for (k = 0; k < UB_RESOURCES; k++)
   ub->ub_parms[k].barrier = ub->ub_parms[k].limit;
 }
--- ./kernel/ub/kmem.c.kmemcore 2006-08-16 19:10:38.000000000 +0400
+++ ./kernel/ub/kmem.c 2006-08-16 19:10:51.000000000 +0400
@@ -0,0 +1,89 @@
+/*
+ *  kernel/ub/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 <ub/beancounter.h>
+#include <ub/kmem.h>
+#include <ub/task.h>
+
+/*
+ * Slab accounting
+ */
+
+int ub_slab_charge(kmem_cache_t *cachep, void *objp, gfp_t flags)
+{
+ unsigned int size;
+ struct user_beancounter *ub, **slab_ubp;
+
+ ub = get_exec_ub();
+ if (ub == NULL)
+  return 0;
+
+ size = kmem_cache_size(cachep);
+ if (charge_beancounter(ub, UB_KMEMSIZE, size,
+    (flags & __GFP_UBC_LIMIT ? UB_LIMIT : UB_BARRIER)))
+  return -ENOMEM;
+
+ slab_ubp = kmem_cache_ubp(cachep, objp);
+ *slab_ubp = get_beancounter(ub);
+ return 0;
+}
+
+void ub_slab_uncharge(kmem_cache_t *cachep, void *objp)
+{
+ unsigned int size;
```

```
+ struct user_beancounter *ub, **slab_ubp;
+
+ slab_ubp = kmem_cache_ubp(cachep, objp);
+ if (*slab_ubp == NULL)
+  return;
+
+ ub = *slab_ubp;
+ size = kmem_cache_size(cachep);
+ uncharge_beancounter(ub, UB_KMEMSIZE, size);
+ put_beancounter(ub);
+ *slab_ubp = NULL;
+}
+
+/*
+ * Pages accounting
+ */
+
+int ub_page_charge(struct page *page, int order, gfp_t flags)
+{
+ struct user_beancounter *ub;
+
+ BUG_ON(page_ub(page) != NULL);
+
+ ub = get_exec_ub();
+ if (ub == NULL)
+  return 0;
+
+ if (charge_beancounter(ub, UB_KMEMSIZE, PAGE_SIZE << order,
+    (flags & __GFP_UBC_LIMIT ? UB_LIMIT : UB_BARRIER)))
+  return -ENOMEM;
+
+ page_ub(page) = get_beancounter(ub);
+ return 0;
+}
+
+void ub_page_uncharge(struct page *page, int order)
+{
+ struct user_beancounter *ub;
+
+ ub = page_ub(page);
+ if (ub == NULL)
+  return;
+
+ uncharge_beancounter(ub, UB_KMEMSIZE, PAGE_SIZE << order);
+ put_beancounter(ub);
+ page_ub(page) = NULL;
+}
--- ./mm/mempool.c.kmemcore 2006-08-16 19:10:38.000000000 +0400
```

```
+++ ./mm/mempool.c 2006-08-16 19:10:51.000000000 +0400
@@ -119,6 +119,7 @@ int mempool_resize(mempool_t *pool, int
  unsigned long flags;

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

  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_UBC;  /* do not charge */

  gfp_temp = gfp_mask & ~(__GFP_WAIT|__GFP_IO);

--- ./mm/page_alloc.c.kmemcore 2006-08-16 19:10:38.000000000 +0400
+++ ./mm/page_alloc.c 2006-08-16 19:10:51.000000000 +0400
@@ -38,6 +38,8 @@
 #include <linux/mempolicy.h>
 #include <linux/stop_machine.h>

+#include <ub/kmem.h>
+
 #include <asm/tlbflush.h>
 #include <asm/div64.h>
 #include "internal.h"
@@ -484,6 +486,8 @@ static void __free_pages_ok(struct page
  if (reserved)
   return;

+ ub_page_uncharge(page, order);
+
  kernel_map_pages(page, 1 << order, 0);
  local_irq_save(flags);
  __count_vm_events(PGFREE, 1 << order);
@@ -764,6 +768,8 @@ static void fastcall free_hot_cold_page(
  if (free_pages_check(page))
   return;

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

  pcp = &zone_pcp(zone, get_cpu())->pcp[cold];
@@ -1153,6 +1159,11 @@ nopage:
   show_mem();
```

```
  }
 got_pg:
+ if ((gfp_mask & __GFP_UBC) &&
+    ub_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.kmemcore 2006-08-16 19:10:38.000000000 +0400
+++ ./mm/slab.c 2006-08-16 19:10:51.000000000 +0400
@@ -108,6 +108,8 @@
 #include <linux/mutex.h>
 #include <linux/rtmutex.h>

+#include <ub/kmem.h>
+
 #include <asm/uaccess.h>
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>
@@ -175,11 +177,13 @@
     SLAB_CACHE_DMA | \
     SLAB_MUST_HWCACHE_ALIGN | SLAB_STORE_USER | \
     SLAB_RECLAIM_ACCOUNT | SLAB_PANIC | \
+    SLAB_UBC | SLAB_UBC_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_UBC | SLAB_UBC_NOCHARGE | \
     SLAB_DESTROY_BY_RCU | SLAB_MEM_SPREAD)
 #endif

@@ -801,9 +805,33 @@ static struct kmem_cache *kmem_find_gene
  return __find_general_cachep(size, gfpflags);
 }

-static size_t slab_mgmt_size(size_t nr_objs, size_t align)
+static size_t slab_mgmt_size_raw(size_t nr_objs)
+{
+ return sizeof(struct slab) + nr_objs * sizeof(kmem_bufctl_t);
+}
+
+#ifdef CONFIG_USER_RESOURCE
+#define UB_EXTRASIZE sizeof(struct user_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_UBC)
+  size = ALIGN(size, UB_EXTRASIZE) + nr_objs * UB_EXTRASIZE;
+ return size;
+}
+#else
+#define UB_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(sizeof(struct slab)+nr_objs*sizeof(kmem_bufctl_t), align);
+ return ALIGN(slab_mgmt_size_noalign(flags, nr_objs), align);
 }

 /*
@@ -848,20 +876,21 @@ 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_UBC ? UB_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;
```

```
@@ -1420,7 +1449,8 @@ 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_UBC |
+       SLAB_UBC_NOCHARGE | SLAB_PANIC,
       NULL, NULL);

  if (INDEX_AC != INDEX_L3) {
@@ -1428,7 +1458,8 @@ 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_UBC |
+      SLAB_UBC_NOCHARGE | SLAB_PANIC,
      NULL, NULL);
  }

@@ -1446,7 +1477,8 @@ 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_UBC |
+       SLAB_UBC_NOCHARGE | SLAB_PANIC,
       NULL, NULL);
    }

@@ -1943,7 +1975,8 @@ 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_UBC ? UB_EXTRASIZE : 0));

     if (num > offslab_limit)
     break;
@@ -2251,8 +2284,8 @@ 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
@@ -2263,11 +2296,9 @@ 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. */
@@ -2513,6 +2544,30 @@ 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_USER_RESOURCE
+static inline struct user_beancounter **slab_ub_ptrs(kmem_cache_t *cachep,
+  struct slab *slabp)
+{
+ return (struct user_beancounter **) ALIGN((unsigned long)
+   (slab_bufctl(slabp) + cachep->num), UB_EXTRASIZE);
+}
+
+struct user_beancounter **kmem_cache_ubp(kmem_cache_t *cachep, void *objp)
+{
+ struct slab *slabp;
+ struct user_beancounter **ubs;
+
+ slabp = virt_to_slab(objp);
+ ubs = slab_ub_ptrs(cachep, slabp);
+ return ubs + 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
```

```
@@ -2533,7 +2588,8 @@ 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_UBC),
+           nodeid);
   if (!slabp)
    return NULL;
  } else {
@@ -2544,14 +2600,14 @@ static struct slab *alloc_slabmgmt(struc
  slabp->colouroff = colour_off;
  slabp->s_mem = objp + colour_off;
  slabp->nodeid = nodeid;
+#ifdef CONFIG_USER_RESOURCE
+ if (cachep->flags & SLAB_UBC)
+  memset(slab_ub_ptrs(cachep, slabp), 0,
+    cachep->num * UB_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)
 {
@@ -2729,7 +2785,7 @@ 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_UBC), nodeid);
  if (!objp)
   goto failed;

@@ -3077,6 +3133,19 @@ static inline void *____cache_alloc(stru
  return objp;
 }

+static inline int ub_should_charge(kmem_cache_t *cachep, gfp_t flags)
+{
+#ifdef CONFIG_USER_RESOURCE
+ if (!(cachep->flags & SLAB_UBC))
+  return 0;
```

```
+ if (flags & __GFP_UBC)
+  return 1;
+ if (!(cachep->flags & SLAB_UBC_NOCHARGE))
+  return 1;
+#endif
+ return 0;
+}
+
 static __always_inline void *__cache_alloc(struct kmem_cache *cachep,
       gfp_t flags, void *caller)
 {
@@ -3090,6 +3159,12 @@ static __always_inline void *__cache_all
  local_irq_restore(save_flags);
  objp = cache_alloc_debugcheck_after(cachep, flags, objp,
         caller);
+
+ if (objp && ub_should_charge(cachep, flags))
+  if (ub_slab_charge(cachep, objp, flags)) {
+   kmem_cache_free(cachep, objp);
+   objp = NULL;
+  }
  prefetchw(objp);
  return objp;
 }
@@ -3287,6 +3362,8 @@ static inline void __cache_free(struct k
  struct array_cache *ac = cpu_cache_get(cachep);

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

  if (cache_free_alien(cachep, objp))
--- ./mm/vmalloc.c.kmemcore 2006-08-16 19:10:38.000000000 +0400
+++ ./mm/vmalloc.c 2006-08-16 19:10:51.000000000 +0400
@@ -520,6 +520,12 @@ void *vmalloc(unsigned long size)
 }
 EXPORT_SYMBOL(vmalloc);

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

Subject: [RFC][PATCH 6/7] UBC: kernel memory acconting (mark objects)
Posted by dev on Wed, 16 Aug 2006 15:40:48 GMT
View Forum Message <> Reply to Message

Mark some kmem caches with SLAB_UBC and some allocations with __GFP_UBC
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.ubslabs 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_ub(mincount*LDT_ENTRY_SIZE);
  else
-  newldt = kmalloc(mincount*LDT_ENTRY_SIZE, GFP_KERNEL);
+  newldt = kmalloc(mincount*LDT_ENTRY_SIZE, GFP_KERNEL_UBC);

  if (!newldt)
   return -ENOMEM;
--- ./arch/i386/mm/init.c.ubslabs 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_UBC,
      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_UBC,
     pgd_ctor,
     PTRS_PER_PMD == 1 ? pgd_dtor : NULL);
  if (!pgd_cache)
--- ./arch/i386/mm/pgtable.c.ubslabs 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_UBC | __GFP_UBC_LIMIT, 0);
 #else
- pte = alloc_pages(GFP_KERNEL|__GFP_REPEAT|__GFP_ZERO, 0);
+ pte = alloc_pages(GFP_KERNEL|__GFP_REPEAT|__GFP_ZERO|
+    __GFP_UBC | __GFP_UBC_LIMIT, 0);
 #endif
  return pte;
 }
--- ./drivers/char/tty_io.c.ubslabs 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
 {
  struct tty_struct *tty;

- tty = kmalloc(sizeof(struct tty_struct), GFP_KERNEL);
+ tty = kmalloc(sizeof(struct tty_struct), GFP_KERNEL_UBC);
  if (tty)
   memset(tty, 0, sizeof(struct tty_struct));
  return tty;
@@ -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_UBC);
   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_UBC);
   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_UBC);
    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_UBC);
    if (!o_ltp)
     goto free_mem_out;
    memset(o_ltp, 0, sizeof(struct termios));
--- ./fs/file.c.ubslabs 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_UBC);
  else
-  new_fds = (struct file **) vmalloc(size);
+  new_fds = (struct file **) vmalloc_ub(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_UBC);
  else
-  new_fdset = (fd_set *) vmalloc(size);
+  new_fdset = (fd_set *) vmalloc_ub(size);
  return new_fdset;
 }

--- ./fs/locks.c.ubslabs 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_UBC,
    init_once, NULL);
  return 0;
 }
--- ./fs/namespace.c.ubslabs 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_UBC | SLAB_PANIC,
+   NULL, NULL);

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

--- ./fs/select.c.ubslabs 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_UBC);
   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_UBC);
   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_UBC);
    if (!pp)
     goto out_fds;
   }
--- ./include/asm-i386/thread_info.h.ubslabs 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_UBC); \
   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_UBC)
 #endif

 #define free_thread_info(info) kfree(info)
--- ./include/asm-ia64/pgalloc.h.ubslabs 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 <ub/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 && ub_page_charge(virt_to_page(ret),
+      0, __GFP_UBC_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_UBC | __GFP_UBC_LIMIT);
  }

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

  preempt_disable();
+ ub_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.ubslabs 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_UBC | __GFP_UBC_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_UBC | __GFP_UBC_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_UBC | __GFP_UBC_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_UBC | __GFP_UBC_LIMIT);
  if (!p)
   return NULL;
  return virt_to_page(p);
--- ./include/asm-x86_64/thread_info.h.ubslabs 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_UBC, \
+    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_UBC,THREAD_ORDER))
 #endif

 #define free_thread_info(ti) free_pages((unsigned long) (ti), THREAD_ORDER)
--- ./ipc/msgutil.c.ubslabs 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_UBC);
  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_UBC);
   if (seg == NULL) {
    err = -ENOMEM;
    goto out_err;
--- ./ipc/sem.c.ubslabs 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_UBC);
   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_UBC);
  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_UBC);
   if(sops==NULL)
    return -ENOMEM;
  }
--- ./ipc/util.c.ubslabs 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_ub(size);
  else
-  out = kmalloc(size, GFP_KERNEL);
+  out = kmalloc(size, GFP_KERNEL_UBC);
  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_ub(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_UBC);
   if (out) {
    out += HDRLEN_KMALLOC;
    container_of(out, struct ipc_rcu_hdr, data)->is_vmalloc = 0;
--- ./kernel/fork.c.ubslabs 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_UBC, 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_UBC,
    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_UBC, 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_UBC, 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_UBC, NULL, NULL);
  vm_area_cachep = kmem_cache_create("vm_area_struct",
    sizeof(struct vm_area_struct), 0,
-   SLAB_PANIC, NULL, NULL);
+   SLAB_PANIC|SLAB_UBC, 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_UBC, NULL, NULL);
 }


--- ./kernel/posix-timers.c.ubslabs 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_UBC,
+     NULL, NULL);
  idr_init(&posix_timers_id);
  return 0;
 }
--- ./kernel/signal.c.ubslabs 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_UBC, NULL, NULL);
 }
--- ./kernel/user.c.ubslabs 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_UBC, NULL, NULL);

  for(n = 0; n < UIDHASH_SZ; ++n)
   INIT_LIST_HEAD(uidhash_table + n);
--- ./mm/rmap.c.ubslabs 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_UBC,
+   anon_vma_ctor, NULL);
 }

 /*
--- ./mm/shmem.c.ubslabs 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_UBC);
   if (page)
    set_page_private(page, 0);
   spin_lock(&info->lock);
```

Subject: [RFC][PATCH 7/7] UBC: proc interface
Posted by dev on Wed, 16 Aug 2006 15:42:28 GMT

View Forum Message <> Reply to Message

Add proc interface (/proc/user_beancounters) allowing to see current
state (usage/limits/fails for each UB). Implemented via seq files.

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

```
 init/main.c       |   1
 kernel/ub/Makefile |   1
 kernel/ub/proc.c   | 205 ++++++++++++++++++++++++++++++++++++++++++++++++++++
 3 files changed, 207 insertions(+)
```

```
--- ./init/main.c.ubproc 2006-07-31 18:40:20.000000000 +0400
+++ ./init/main.c 2006-08-03 16:02:19.000000000 +0400
@@ -578,6 +578,7 @@ asmlinkage void __init start_kernel(void
  page_writeback_init();
 #ifdef CONFIG_PROC_FS
  proc_root_init();
+ ub_init_proc();
 #endif
  cpuset_init();
  taskstats_init_early();
--- ./kernel/ub/Makefile.ubproc 2006-07-31 17:49:05.000000000 +0400
+++ ./kernel/ub/Makefile 2006-08-01 11:08:39.000000000 +0400
@@ -4,3 +4,4 @@ obj-$(CONFIG_USER_RESOURCE) += beancount
 obj-$(CONFIG_USER_RESOURCE) += misc.o
 obj-y += sys.o
 obj-$(CONFIG_USER_RESOURCE) += kmem.o
+obj-$(CONFIG_USER_RESOURCE) += proc.o
--- ./kernel/ub/proc.c.ubproc 2006-08-01 10:22:09.000000000 +0400
+++ ./kernel/ub/proc.c 2006-08-03 15:50:35.000000000 +0400
@@ -0,0 +1,205 @@
+/*
+ *  kernel/ub/proc.c
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc.
+ *
+ */
+
+#include <linux/sched.h>
+#include <linux/kernel.h>
+#include <linux/proc_fs.h>
+#include <linux/seq_file.h>
+
+#include <ub/beancounter.h>
+
+#ifdef CONFIG_PROC_FS
+
+#if BITS_PER_LONG == 32
+static const char *head_fmt = "%10s %-12s %10s %10s %10s %10s %10s\n";
+static const char *res_fmt = "%10s %-12s %10lu %10lu %10lu %10lu %10lu\n";
+#else
+static const char *head_fmt = "%10s %-12s %20s %20s %20s %20s %20s\n";
+static const char *res_fmt = "%10s %-12s %20lu %20lu %20lu %20lu %20lu\n";
+#endif
+
+static void ub_show_header(struct seq_file *f)
+{
+ seq_printf(f, head_fmt, "uid", "resource",
```

```
+   "held", "maxheld", "barrier", "limit", "failcnt");
+}
+
+static void ub_show_res(struct seq_file *f, struct user_beancounter *ub, int r)
+{
+ char ub_uid[64];
+
+ if (r == 0)
+  ub_print_uid(ub, ub_uid, sizeof(ub_uid));
+ else
+  strcpy(ub_uid, "");
+
+ seq_printf(f, res_fmt, ub_uid, ub_rnames[r],
+   ub->ub_parms[r].held,
+   ub->ub_parms[r].maxheld,
+   ub->ub_parms[r].barrier,
+   ub->ub_parms[r].limit,
+   ub->ub_parms[r].failcnt);
+}
+
+static struct ub_seq_struct {
+ unsigned long flags;
+ int slot;
+ struct user_beancounter *ub;
+} ub_seq_ctx;
+
+static int ub_show(struct seq_file *f, void *v)
+{
+ int res;
+
+ for (res = 0; res < UB_RESOURCES; res++)
+  ub_show_res(f, ub_seq_ctx.ub, res);
+ return 0;
+}
+
+static void *ub_start_ctx(struct seq_file *f, unsigned long p, int sub)
+{
+ struct user_beancounter *ub;
+ struct hlist_node *pos;
+ unsigned long flags;
+ int slot;
+
+ if (p == 0)
+  ub_show_header(f);
+
+ spin_lock_irqsave(&ub_hash_lock, flags);
+ ub_seq_ctx.flags = flags;
+
```

```
+ for (slot = 0; slot < UB_HASH_SIZE; slot++)
+  hlist_for_each_entry (ub, pos, &ub_hash[slot], hash) {
+   if (!sub && ub->parent != NULL)
+    continue;
+
+   if (p-- == 0) {
+    ub_seq_ctx.ub = ub;
+    ub_seq_ctx.slot = slot;
+    return &ub_seq_ctx;
+   }
+  }
+
+ return NULL;
+}
+
+static void *ub_next_ctx(struct seq_file *f, loff_t *ppos, int sub)
+{
+ struct user_beancounter *ub;
+ struct hlist_node *pos;
+ int slot;
+
+ ub = ub_seq_ctx.ub;
+
+ pos = &ub->hash;
+ hlist_for_each_entry_continue (ub, pos, hash) {
+  if (!sub && ub->parent != NULL)
+   continue;
+
+  ub_seq_ctx.ub = ub;
+  (*ppos)++;
+  return &ub_seq_ctx;
+ }
+
+ for (slot = ub_seq_ctx.slot + 1; slot < UB_HASH_SIZE; slot++)
+  hlist_for_each_entry (ub, pos, &ub_hash[slot], hash) {
+   if (!sub && ub->parent != NULL)
+    continue;
+
+   ub_seq_ctx.ub = ub;
+   ub_seq_ctx.slot = slot;
+   (*ppos)++;
+   return &ub_seq_ctx;
+  }
+
+ return NULL;
+}
+
+static void *ub_start(struct seq_file *f, loff_t *ppos)
```

```
+{
+ return ub_start_ctx(f, *ppos, 0);
+}
+
+static void *ub_sub_start(struct seq_file *f, loff_t *ppos)
+{
+ return ub_start_ctx(f, *ppos, 1);
+}
+
+static void *ub_next(struct seq_file *f, void *v, loff_t *pos)
+{
+ return ub_next_ctx(f, pos, 0);
+}
+
+static void *ub_sub_next(struct seq_file *f, void *v, loff_t *pos)
+{
+ return ub_next_ctx(f, pos, 1);
+}
+
+static void ub_stop(struct seq_file *f, void *v)
+{
+ unsigned long flags;
+
+ flags = ub_seq_ctx.flags;
+ spin_unlock_irqrestore(&ub_hash_lock, flags);
+}
+
+static struct seq_operations ub_seq_ops = {
+ .start = ub_start,
+ .next  = ub_next,
+ .stop  = ub_stop,
+ .show  = ub_show
+};
+
+static int ub_open(struct inode *inode, struct file *filp)
+{
+ return seq_open(filp, &ub_seq_ops);
+}
+
+static struct file_operations ub_file_operations = {
+ .open  = ub_open,
+ .read  = seq_read,
+ .llseek  = seq_lseek,
+ .release = seq_release,
+};
+
+static struct seq_operations ub_sub_seq_ops = {
+ .start = ub_sub_start,
```

```
+ .next  = ub_sub_next,
+ .stop  = ub_stop,
+ .show  = ub_show
+};
+
+static int ub_sub_open(struct inode *inode, struct file *filp)
+{
+ return seq_open(filp, &ub_sub_seq_ops);
+}
+
+static struct file_operations ub_sub_file_operations = {
+ .open  = ub_sub_open,
+ .read  = seq_read,
+ .llseek  = seq_lseek,
+ .release = seq_release,
+};
+
+void __init ub_init_proc(void)
+{
+ struct proc_dir_entry *entry;
+
+ entry = create_proc_entry("user_beancounters", S_IRUGO, NULL);
+ if (entry)
+  entry->proc_fops = &ub_file_operations;
+ else
+  panic("Can't create /proc/user_beancounters\n");
+
+ entry = create_proc_entry("user_beancounters_sub", S_IRUGO, NULL);
+ if (entry)
+  entry->proc_fops = &ub_sub_file_operations;
+ else
+  panic("Can't create /proc/user_beancounters_sub\n");
+}
+#endif
```

Subject: Re: [RFC][PATCH 3/7] UBC: ub context and inheritance
Posted by Alan Cox on Wed, 16 Aug 2006 16:31:01 GMT
View Forum Message <> Reply to Message

Ar Mer, 2006-08-16 am 19:38 +0400, ysgrifennodd Kirill Korotaev:
> Contains code responsible for setting UB on task,
> it's inheriting and setting host context in interrupts.
>
> Task references three beancounters:
>   1. exec_ub current context. all resources are
>   charged to this beancounter.
>   2. task_ub beancounter to which task_struct is

> charged itself.
> 3. fork_sub 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>

Acked-by: Alan Cox <alan@redhat.com>

Subject: Re: [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by Alan Cox on Wed, 16 Aug 2006 16:32:17 GMT
View Forum Message <> Reply to Message

Ar Mer, 2006-08-16 am 19:39 +0400, ysgrifennodd Kirill Korotaev:
> Add the following system calls for UB management:
> 1. sys_getluid - get current UB id
> 2. sys_setluid - changes exec_ and fork_ UBs on current
> 3. sys_setublimit - set limits for resources consumtions
>
> Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
> Signed-Off-By: Kirill Korotaev <dev@sw.ru>

Acked-by: Alan Cox <alan@redhat.com>

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Alan Cox on Wed, 16 Aug 2006 16:35:31 GMT
View Forum Message <> Reply to Message

The

+ ub->ub_parms[UB_KMEMSIZE].limit = 32 * 1024 * 1024

seems a bit arbitary. 32Mb is variously vast amounts of memory and not
enough to boot depending if you are booting a PDA or a 4096 core Itanic
box

Subject: Re: [RFC][PATCH 6/7] UBC: kernel memory acconting (mark objects)
Posted by Alan Cox on Wed, 16 Aug 2006 16:36:59 GMT
View Forum Message <> Reply to Message

Ar Mer, 2006-08-16 am 19:42 +0400, ysgrifennodd Kirill Korotaev:
> Mark some kmem caches with SLAB_UBC and some allocations with __GFP_UBC
> to cause charging/limiting of appropriate kernel resources.
>

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

Acked-by: Alan Cox <alan@redhat.com>

(although it will clash slightly with the diffs I just sent Andrew)

Subject: Re: [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Alan Cox on Wed, 16 Aug 2006 16:38:16 GMT
View Forum Message <> Reply to Message

Ar Mer, 2006-08-16 am 19:37 +0400, ysgrifennodd Kirill Korotaev:
> + * UB_MAXVALUE is essentially LONG_MAX declared in a cross-compiling safe form.
> + */
> +#define UB_MAXVALUE ((1UL << (sizeof(unsigned long)*8-1)) - 1)
> +

Whats wrong with using the kernels LONG_MAX ?

Subject: Re: [RFC][PATCH 7/7] UBC: proc interface
Posted by Greg KH on Wed, 16 Aug 2006 17:13:28 GMT
View Forum Message <> Reply to Message

On Wed, Aug 16, 2006 at 07:44:30PM +0400, Kirill Korotaev wrote:
> Add proc interface (/proc/user_beancounters) allowing to see current
> state (usage/limits/fails for each UB). Implemented via seq files.

Ugh, why /proc? This doesn't have anything to do with processes, just
users, right? What's wrong with /sys/kernel/ instead?

Or /sys/kernel/debug/user_beancounters/ in debugfs as this is just a
debugging thing, right?

thanks,

greg k-h

Subject: Re: [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Greg KH on Wed, 16 Aug 2006 17:15:27 GMT
View Forum Message <> Reply to Message

On Wed, Aug 16, 2006 at 07:37:26PM +0400, Kirill Korotaev wrote:
> +struct user_beancounter
> +{

> + atomic_t ub_refcount;

Why not use a struct kref here instead of rolling your own reference
counting logic?

thanks,

greg k-h

Subject: Re: [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by Greg KH on Wed, 16 Aug 2006 17:17:47 GMT
View Forum Message <> Reply to Message

On Wed, Aug 16, 2006 at 07:39:43PM +0400, Kirill Korotaev wrote:
> --- ./include/asm-sparc/unistd.h.arsys 2006-07-10 12:39:19.000000000 +0400
> +++ ./include/asm-sparc/unistd.h 2006-08-10 17:08:19.000000000 +0400
> @@ -318,6 +318,9 @@
> #define __NR_unshare 299
> #define __NR_set_robust_list 300
> #define __NR_get_robust_list 301
> +#define __NR_getluid 302
> +#define __NR_setluid 303
> +#define __NR_setublimit 304

Hm, you seem to be ignoring this:

>
> #ifdef __KERNEL__
> /* WARNING: You MAY NOT add syscall numbers larger than 301, since

Same thing for sparc64:

> --- ./include/asm-sparc64/unistd.h.arsys 2006-07-10
> 12:39:19.000000000 +0400
> +++ ./include/asm-sparc64/unistd.h 2006-08-10 17:09:24.000000000 +0400
> @@ -320,6 +320,9 @@
> #define __NR_unshare 299
> #define __NR_set_robust_list 300
> #define __NR_get_robust_list 301
> +#define __NR_getluid 302
> +#define __NR_setluid 303
> +#define __NR_setublimit 304
>
> #ifdef __KERNEL__
> /* WARNING: You MAY NOT add syscall numbers larger than 301, since

You might want to read those comments...

thanks,

greg k-h

Subject: Re: [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Rohit Seth on Wed, 16 Aug 2006 18:11:08 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-16 at 19:37 +0400, Kirill Korotaev wrote:
> Core functionality and interfaces of UBC:
> find/create beancounter, initialization,
> charge/uncharge of resource, core objects' declarations.
>
> Basic structures:
> ubparm - resource description
> user_beancounter - set of resources, id, lock
>
> Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
> Signed-Off-By: Kirill Korotaev <dev@sw.ru>
>
> ---
> include/ub/beancounter.h | 157 ++++++++++++++++++
> init/main.c | 4
> kernel/Makefile | 1
> kernel/ub/Makefile | 7
> kernel/ub/beancounter.c | 398 ++
> 5 files changed, 567 insertions(+)
>
> --- /dev/null 2006-07-18 14:52:43.075228448 +0400
> +++ ./include/ub/beancounter.h 2006-08-10 14:58:27.000000000 +0400
> @@ -0,0 +1,157 @@
> +/*
> + * include/ub/beancounter.h
> + *
> + * Copyright (C) 2006 OpenVZ. SWsoft Inc
> + *
> + */
> +
> +#ifndef _LINUX_BEANCOUNTER_H
> +#define _LINUX_BEANCOUNTER_H
> +
> +/*
> + * Resource list.
> + */
> +
> +#define UB_RESOURCES 0

> +
> +struct ubparm {
> + /*
> + * 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 barrier;
> + /* hard resource limit */
> + unsigned long limit;
> + /* consumed resources */
> + unsigned long held;
> + /* maximum amount of consumed resources through the last period */
> + unsigned long maxheld;
> + /* minimum amount of consumed resources through the last period */
> + unsigned long minheld;
> + /* count of failed charges */
> + unsigned long failcnt;
> +};

What is the difference between barrier and limit. They both sound like
hard limits. No?

> +
> +/*
> + * Kernel internal part.
> + */
> +
> +#ifdef __KERNEL__
> +
> +#include <linux/config.h>
> +#include <linux/spinlock.h>
> +#include <linux/list.h>
> +#include <asm/atomic.h>
> +
> +/*
> + * UB_MAXVALUE is essentially LONG_MAX declared in a cross-compiling safe form.
> + */
> +#define UB_MAXVALUE ((1UL << (sizeof(unsigned long)*8-1)) - 1)
> +
> +
> +/*
> + * Resource management structures
> + * Serialization issues:
> + *   beancounter list management is protected via ub_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-ub spinlock

> + */
> +
> +struct user_beancounter
> +{
> + atomic_t ub_refcount;
> + spinlock_t ub_lock;
> + uid_t   ub_uid;

Why uid? Will it be possible to club processes belonging to different
users to same bean counter.

> + struct hlist_node hash;
> +
> + struct user_beancounter *parent;
> + void   *private_data;
> +

What are the above two fields used for?

> + /* resources statistics and settings */
> + struct ubparm ub_parms[UB_RESOURCES];
> +};
> +

I presume UB_RESOURCES value is going to change as different resources
start getting tracked.

I think something like configfs should be used for user interface. It
automatically presents the right interfaces to user land (based on
kernel implementation). And you wouldn't need any changes in glibc etc.

-rohit

Subject: Re: [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by Rohit Seth on Wed, 16 Aug 2006 18:17:57 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-16 at 19:39 +0400, Kirill Korotaev wrote:
> Add the following system calls for UB management:
>   1. sys_getluid   - get current UB id
>   2. sys_setluid   - changes exec_ and fork_ UBs on current
>   3. sys_setublimit - set limits for resources consumtions
>

Why not have another system call for getting the current limits?

But as I said in previous mail, configfs seems like a better choice for
user interface. That way user has to go to one place to read/write
limits, see the current usage and other stats.

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

 ...<snip>...
> +
> +/*
> + * The setbeanlimit syscall
> + */
> +asmlinkage long sys_setublimit(uid_t uid, unsigned long resource,
> + unsigned long *limits)
> +{

> + ub = beancounter_findcreate(uid, NULL, 0);
> + if (ub == NULL)
> + goto out;
> +
> + spin_lock_irqsave(&ub->ub_lock, flags);
> + ub->ub_parms[resource].barrier = new_limits[0];
> + ub->ub_parms[resource].limit = new_limits[1];
> + spin_unlock_irqrestore(&ub->ub_lock, flags);
> +

I think there should be a check here for seeing if the new limits are
lower than the current usage of a resource. If so then take appropriate
action.

-rohit

Subject: Re: [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Andrew Morton on Wed, 16 Aug 2006 18:18:18 GMT
View Forum Message <> Reply to Message

On Wed, 16 Aug 2006 11:11:08 -0700
Rohit Seth <rohitseth@google.com> wrote:

> > +struct user_beancounter
> > +{
> > + atomic_t ub_refcount;
> > + spinlock_t ub_lock;
> > + uid_t   ub_uid;
>
> Why uid? Will it be possible to club processes belonging to different
> users to same bean counter.

hm. I'd have expected to see a `struct user_struct *' here, not a uid_t.

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Rohit Seth on Wed, 16 Aug 2006 18:24:53 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-16 at 19:40 +0400, Kirill Korotaev wrote:
> Introduce UB_KMEMSIZE resource which accounts kernel
> objects allocated by task's request.
>
> Reference to UB 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_UBC flag - page
>   is charged to current's exec_ub.
> 2. Slabs - kmem_cache may be created with SLAB_UBC flag - in this
>   case each allocation is charged. Caches used by kmalloc are
>   created with SLAB_UBC | SLAB_UBC_NOCHARGE flags. In this case
>   only __GFP_UBC allocations are charged.

<snip>

> --- ./mm/page_alloc.c.kmemcore 2006-08-16 19:10:38.000000000 +0400
> +++ ./mm/page_alloc.c 2006-08-16 19:10:51.000000000 +0400
> @@ -38,6 +38,8 @@
> #include <linux/mempolicy.h>
> #include <linux/stop_machine.h>
>
> +#include <ub/kmem.h>
> +
> #include <asm/tlbflush.h>
> #include <asm/div64.h>
> #include "internal.h"
> @@ -484,6 +486,8 @@ static void __free_pages_ok(struct page
>   if (reserved)
>   return;
>
> + ub_page_uncharge(page, order);
> +
>   kernel_map_pages(page, 1 << order, 0);
>   local_irq_save(flags);
>   __count_vm_events(PGFREE, 1 << order);
> @@ -764,6 +768,8 @@ static void fastcall free_hot_cold_page(
>   if (free_pages_check(page))

> return;
>
> + ub_page_uncharge(page, 0);
> +
> kernel_map_pages(page, 1, 0);
>
> pcp = &zone_pcp(zone, get_cpu())->pcp[cold];
> @@ -1153,6 +1159,11 @@ nopage:
> show_mem();
> }
> got_pg:
> + if ((gfp_mask & __GFP_UBC) &&
> + ub_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);

If I'm reading this patch right then seems like you are making page allocations to fail w/o (for example) trying to purge some pages from the page cache belonging to this container. Or is that reclaim going to come later?

-rohit

Subject: Re: [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by Alan Cox on Wed, 16 Aug 2006 18:44:44 GMT
View Forum Message <> Reply to Message

Ar Mer, 2006-08-16 am 11:17 -0700, ysgrifennodd Rohit Seth:
> I think there should be a check here for seeing if the new limits are
> lower than the current usage of a resource. If so then take appropriate
> action.

Generally speaking there isn't a sane appropriate action because the resources can't just be yanked.

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Dave Hansen on Wed, 16 Aug 2006 18:47:09 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-16 at 19:40 +0400, Kirill Korotaev wrote:
> --- ./include/linux/mm.h.kmemcore	2006-08-16 19:10:38.000000000

> +0400
> +++ ./include/linux/mm.h 2006-08-16 19:10:51.000000000 +0400
> @@ -274,8 +274,14 @@ struct page {
> unsigned int gfp_mask;
> unsigned long trace[8];
> #endif
> +#ifdef CONFIG_USER_RESOURCE
> + union {
> + struct user_beancounter *page_ub;
> + } bc;
> +#endif
> };

Is everybody OK with adding this accounting to the 'struct page'? Is there any kind of noticeable performance penalty for this? I thought that we had this aligned pretty well on cacheline boundaries.

How many things actually use this? Can we have the slab ubcs without the struct page pointer?

-- Dave

Subject: Re: [RFC][PATCH] UBC: user resource beancounters
Posted by Rohit Seth on Wed, 16 Aug 2006 18:53:47 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-16 at 19:24 +0400, Kirill Korotaev wrote:
> The following patch set presents base of
> User Resource Beancounters (UBC).
> UBC allows to account and control consumption
> of kernel resources used by group of processes.
>
> The full UBC 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.
>

Good.

> - virtual memory pages. UBC allows 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).

>

I wouldn't be too worried about doing fractions. Make it unfair and
charge it to either the container who first instantiated the file or the
container who faulted on that page first.

Though the part that seems important is to be able to define a directory
in fs and say all pages belonging to files underneath that directory are
going to be put in specific container. Just like you are having
resource beans associated with sockets, have address_space or inode also
associated with resource beans. (And it should be possible to have a
container/resource bean without any active process but set of
address_space mappings with its own limits and current usage).

-rohit

Subject: Re: [RFC][PATCH] UBC: user resource beancounters
Posted by Alan Cox on Wed, 16 Aug 2006 19:06:25 GMT
View Forum Message <> Reply to Message

Ar Mer, 2006-08-16 am 11:53 -0700, ysgrifennodd Rohit Seth:
> >   pages shared between containers are correctly
> >   charged as fractions (tunable).
> >
>
> I wouldn't be too worried about doing fractions. Make it unfair and
> charge it to either the container who first instantiated the file or the
> container who faulted on that page first.

Thats no good if you can arrange who gets charged, it becomes possible
to accumulate the advantages and break the constraints intended.

> Though the part that seems important is to be able to define a directory
> in fs and say all pages belonging to files underneath that directory are
> going to be put in specific container.

Thats an extremely crude use of beancounters. You can do far more useful
things with them and namespaces (and even at times without namespaces)
such as preventing one web site breaking another.

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Rohit Seth on Wed, 16 Aug 2006 19:15:29 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-16 at 11:47 -0700, Dave Hansen wrote:

> On Wed, 2006-08-16 at 19:40 +0400, Kirill Korotaev wrote:
> > --- ./include/linux/mm.h.kmemcore 2006-08-16 19:10:38.000000000
> > +0400
> > +++ ./include/linux/mm.h 2006-08-16 19:10:51.000000000 +0400
> > @@ -274,8 +274,14 @@ struct page {
> > unsigned int gfp_mask;
> > unsigned long trace[8];
> > #endif
> > +#ifdef CONFIG_USER_RESOURCE
> > + union {
> > + struct user_beancounter *page_ub;
> > + } bc;
> > +#endif
> > };
>
> Is everybody OK with adding this accounting to the 'struct page'?

My preference would be to have container (I keep on saying container, but resource beancounter) pointer embeded in task, mm(not sure), address_space and anon_vma structures. This should allow us to track user land pages optimally. But for tracking kernel usage on behalf of user, we will have to use an additional field (unless we can re-use mapping). Please correct me if I'm wrong, though all the kernel resources will be allocated/freed in context of a user process. And at that time we know if a allocation should succeed or not. So we may actually not need to track kernel pages that closely. We are not going to run reclaim on any of them anyways.

-rohit

Subject: Re: [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by Rohit Seth on Wed, 16 Aug 2006 19:22:50 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-16 at 20:04 +0100, Alan Cox wrote:
> Ar Mer, 2006-08-16 am 11:17 -0700, ysgrifennodd Rohit Seth:
> > I think there should be a check here for seeing if the new limits are
> > lower than the current usage of a resource. If so then take appropriate
> > action.
>
> Generally speaking there isn't a sane appropriate action because the
> resources can't just be yanked.
>

I was more thinking about (for example) user land physical memory limit for that bean counter. If the limits are going down, then the system call should try to flush out page cache pages or swap out anonymous

memory. But you are right that it won't be possible in all cases, like
for in kernel memory limits.

-rohit

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Alan Cox on Thu, 17 Aug 2006 00:02:08 GMT
View Forum Message <> Reply to Message

Ar Mer, 2006-08-16 am 12:15 -0700, ysgrifennodd Rohit Seth:
> resources will be allocated/freed in context of a user process. And at
> that time we know if a allocation should succeed or not. So we may
> actually not need to track kernel pages that closely.

Quite the reverse, tracking kernel pages is critical, user pages kind of
balance out for many cases kernel ones don't.

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Chandra Seetharaman on Thu, 17 Aug 2006 00:15:47 GMT
View Forum Message <> Reply to Message

Kirill,

Thanks for posting the patches to ckrm-tech. I 'll look into it and post
my comments tomorrow.

Some documentation (or pointer to the documentation) on how to use this
feature and high level design would really help.

How does the hierarchy work ? (May be reading the code would clear it
up :).

few comments below..
On Wed, 2006-08-16 at 19:24 +0400, Kirill Korotaev wrote:
<snip>
> The patches in these series are:
> diff-ubc-kconfig.patch:
>   Adds kernel/ub/Kconfig file with UBC options and
>   includes it into arch Kconfigs

Since the core functionality is arch independent, why not have the
Kconfig stuff in some generic place like init/Kconfig ?

>
> diff-ubc-core.patch:

>   Contains core functionality and interfaces of UBC:
>   find/create beancounter, initialization,
>   charge/uncharge of resource, core objects' declarations.
>
> diff-ubc-task.patch:
>   Contains code responsible for setting UB on task,
>   it's inheriting and setting host context in interrupts.
>
>   Task contains three beancounters:
>   1. exec_ub - current context. all resources are charged
>   to this beancounter.
>   2. task_ub - beancounter to which task_struct is charged
>   itself.
>   3. fork_sub - beancounter which is inherited by
>   task's children on fork

wondering why we need three of these ?

>
> diff-ubc-syscalls.patch:
>   Patch adds system calls for UB management:
>   1. sys_getluid   - get current UB id
>   2. sys_setluid   - changes exec_ and fork_ UBs on current
>   3. sys_setublimit - set limits for resources consumtions

I agree with Rohit that configfs based interface would be more easy to
use (you will not get into the system call number issue that Greg has
pointed too).
>
> diff-ubc-kmem-core.patch:
>   Introduces UB_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_UBC flag - page
>   is charged to current's exec_ub.
>   2. Slabs - kmem_cache may be created with SLAB_UBC flag - in this
>   case each allocation is charged. Caches used by kmalloc are
>   created with SLAB_UBC | SLAB_UBC_NOCHARGE flags. In this case
>   only __GFP_UBC allocations are charged.
>
> diff-ubc-kmem-charge.patch:
>   Adds SLAB_UBC and __GFP_UBC flags in appropriate places
>   to cause charging/limiting of specified resources.

>
> diff-ubc-proc.patch:
> Adds two proc entries user_beancounters and user_beancounters_sub
> allowing to see current state (usage/limits/fails for each UB).
> Implemented via seq files.

again, configfs would be easier.

>
> Patch set is applicable to 2.6.18-rc4-mm1
>
> Thanks,
> Kirill
>
>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Srivatsa Vaddagiri on Thu, 17 Aug 2006 11:02:37 GMT
View Forum Message <> Reply to Message

On Wed, Aug 16, 2006 at 07:24:03PM +0400, Kirill Korotaev wrote:
> As the first step we want to propose for discussion
> the most complicated parts of resource management:
> kernel memory and virtual memory.

Do you have any plans to post a CPU controller? Is that tied to UBC
interface as well?

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Srivatsa Vaddagiri on Thu, 17 Aug 2006 11:09:13 GMT
View Forum Message <> Reply to Message

On Wed, Aug 16, 2006 at 07:37:26PM +0400, Kirill Korotaev wrote:
> +struct user_beancounter
> +{
> + atomic_t ub_refcount;
> + spinlock_t ub_lock;
> + uid_t   ub_uid;
> + struct hlist_node hash;
> +
> + struct user_beancounter *parent;

This seems to hint at some heirarchy of ubc? How would that heirarchy be used? I cant find anything in the patch which forms this heirarchy (basically I dont see any place where beancounter_findcreate() is called with non-NULL 2nd arg).

[snip]

> +static void init_beancounter_syslimits(struct user_beancounter *ub)
> +{
> + int k;
> +
> + for (k = 0; k < UB_RESOURCES; k++)
> + ub->ub_parms[k].barrier = ub->ub_parms[k].limit;

This sets barrier to 0. Is this value of 0 interpreted differently by different controllers? One way to interpret it is "dont allocate any resource", other way to interpret it is "don't care - give me what you can" (which makes sense for stuff like CPU and network bandwidth).

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by Srivatsa Vaddagiri on Thu, 17 Aug 2006 11:09:40 GMT
View Forum Message <> Reply to Message

On Wed, Aug 16, 2006 at 07:39:43PM +0400, Kirill Korotaev wrote:

> +/*
> + * The setbeanlimit syscall

> + */
> +asmlinkage long sys_setublimit(uid_t uid, unsigned long resource,
> + unsigned long *limits)
> +{

[snip]

> + spin_lock_irqsave(&ub->ub_lock, flags);
> + ub->ub_parms[resource].barrier = new_limits[0];
> + ub->ub_parms[resource].limit = new_limits[1];

Would it be usefull to notify the "resource" controller about this
change in limits? For ex: in case of the CPU controller I wrote
(http://lkml.org/lkml/2006/8/4/9), I was finding it usefull to recv
notification of changes to these limits, so that internal structures
(which are kept per-task-group) can be updated.

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [RFC][PATCH 3/7] UBC: ub context and inheritance
Posted by Srivatsa Vaddagiri on Thu, 17 Aug 2006 11:09:52 GMT
View Forum Message <> Reply to Message

On Wed, Aug 16, 2006 at 07:38:44PM +0400, Kirill Korotaev wrote:
> Contains code responsible for setting UB on task,
> it's inheriting and setting host context in interrupts.
>
> Task references three beancounters:
> 1. exec_ub current context. all resources are
> charged to this beancounter.
> 2. task_ub beancounter to which task_struct is
> charged itself.
> 3. fork_sub beancounter which is inherited by
> task's children on fork

Is there a case where exec_ub and fork_sub can differ? I dont see that
in the patch.

--
Regards,
vatsa

Subject: Re: [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by dev on Thu, 17 Aug 2006 11:40:51 GMT
View Forum Message <> Reply to Message

> Ar Mer, 2006-08-16 am 19:37 +0400, ysgrifennodd Kirill Korotaev:
>
>>+ * UB_MAXVALUE is essentially LONG_MAX declared in a cross-compiling safe form.
>>+ */
>>+#define UB_MAXVALUE ((1UL << (sizeof(unsigned long)*8-1)) - 1)
>>+
>
>
> Whats wrong with using the kernels LONG_MAX ?
just historical code line which introduces UB_MAXVALUE independant of
cross-compiler/headers etc.

Will replace it.

Kirill

Subject: Re: [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by dev on Thu, 17 Aug 2006 11:43:51 GMT
View Forum Message <> Reply to Message

>>+struct user_beancounter
>>+{
>>+ atomic_t ub_refcount;
>
>
> Why not use a struct kref here instead of rolling your own reference
> counting logic?

We need more complex decrement/locking scheme than krefs
provide. e.g. in __put_beancounter() we need
atomic_dec_and_lock_irqsave() semantics for performance optimizations.

Kirill

Subject: Re: [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by dev on Thu, 17 Aug 2006 11:52:37 GMT
View Forum Message <> Reply to Message

Rohit Seth wrote:
> On Wed, 2006-08-16 at 19:37 +0400, Kirill Korotaev wrote:
>
>>Core functionality and interfaces of UBC:

>>find/create beancounter, initialization,
>>charge/uncharge of resource, core objects' declarations.
>>
>>Basic structures:
>> ubparm   - resource description
>> user_beancounter - set of resources, id, lock
>>
>>Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
>>Signed-Off-By: Kirill Korotaev <dev@sw.ru>
>>
>>---
>> include/ub/beancounter.h | 157 ++++++++++++++++++
>> init/main.c | 4
>> kernel/Makefile | 1
>> kernel/ub/Makefile | 7
>> kernel/ub/beancounter.c | 398 ++
>> 5 files changed, 567 insertions(+)
>>
>>--- /dev/null 2006-07-18 14:52:43.075228448 +0400
>>+++ ./include/ub/beancounter.h 2006-08-10 14:58:27.000000000 +0400
>>@@ -0,0 +1,157 @@
>>+/*
>>+ * include/ub/beancounter.h
>>+ *
>>+ * Copyright (C) 2006 OpenVZ. SWsoft Inc
>>+ *
>>+ */
>>+
>>+#ifndef _LINUX_BEANCOUNTER_H
>>+#define _LINUX_BEANCOUNTER_H
>>+
>>+/*
>>+ * Resource list.
>>+ */
>>+
>>+#define UB_RESOURCES 0
>>+
>>+struct ubparm {
>>+ /*
>>+ * 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 barrier;
>>+ /* hard resource limit */
>>+ unsigned long limit;
>>+ /* consumed resources */
>>+ unsigned long held;

>>+ /* maximum amount of consumed resources through the last period */
>>+ unsigned long maxheld;
>>+ /* minimum amount of consumed resources through the last period */
>>+ unsigned long minheld;
>>+ /* count of failed charges */
>>+ unsigned long failcnt;
>>+};
>
>
> What is the difference between barrier and limit. They both sound like
> hard limits. No?
check __charge_beancounter_locked and severity.
It provides some kind of soft and hard limits.

>>+
>>+/*
>>+ * Kernel internal part.
>>+ */
>>+
>>+#ifdef __KERNEL__
>>+
>>+#include <linux/config.h>
>>+#include <linux/spinlock.h>
>>+#include <linux/list.h>
>>+#include <asm/atomic.h>
>>+
>>+/*
>>+ * UB_MAXVALUE is essentially LONG_MAX declared in a cross-compiling safe form.
>>+ */
>>+#define UB_MAXVALUE ((1UL << (sizeof(unsigned long)*8-1)) - 1)
>>+
>>+
>>+/*
>>+ * Resource management structures
>>+ * Serialization issues:
>>+ *   beancounter list management is protected via ub_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-ub spinlock
>>+ */
>>+
>>+struct user_beancounter
>>+{
>>+ atomic_t ub_refcount;
>>+ spinlock_t ub_lock;
>>+ uid_t   ub_uid;
>
>

> Why uid? Will it be possible to club processes belonging to different
> users to same bean counter.
oh, its a misname. Should be ub_id. it is ID of user_beancounter
and has nothing to do with user id.

>>+ struct hlist_node hash;
>>+
>>+ struct user_beancounter *parent;
>>+ void   *private_data;
>>+
>
>
> What are the above two fields used for?
the first one is for hierarchical UBs,
see beancounter_findcreate with UB_LOOKUP_SUB.
private_data is probably not used yet :)

>>+ /* resources statistics and settings */
>>+ struct ubparm ub_parms[UB_RESOURCES];
>>+};
>>+
>
>
> I presume UB_RESOURCES value is going to change as different resources
> start getting tracked.
what's wrong with it?

> I think something like configfs should be used for user interface. It
> automatically presents the right interfaces to user land (based on
> kernel implementation). And you wouldn't need any changes in glibc etc.
1. UBC doesn't require glibc modificatins.
2. if you think a bit more about it, adding UB parameters doesn't
   require user space changes as well.
3. it is possible to add any kind of interface for UBC. but do you like the idea
   to grep 200(containers)x20(parameters) files for getting current usages?
   Do you like the idea to convert numbers to strings and back w/o
   thinking of data types?

Thanks,
Kirill

Subject: Re: [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by dev on Thu, 17 Aug 2006 11:52:59 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:
> On Wed, 16 Aug 2006 11:11:08 -0700

> Rohit Seth <rohitseth@google.com> wrote:
>
>
>>>+struct user_beancounter
>>>+{
>>>+ atomic_t ub_refcount;
>>>+ spinlock_t ub_lock;
>>>+ uid_t   ub_uid;
>>
>>Why uid? Will it be possible to club processes belonging to different
>>users to same bean counter.
>
>
> hm. I'd have expected to see a `struct user_struct *' here, not a uid_t.

Sorry, misused name. should be ub_id. not related to user_struct or user.

Thanks,
Kirill

Subject: Re: [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by dev on Thu, 17 Aug 2006 12:00:11 GMT
View Forum Message <> Reply to Message

Greg KH wrote:
> On Wed, Aug 16, 2006 at 07:39:43PM +0400, Kirill Korotaev wrote:
>
>>--- ./include/asm-sparc/unistd.h.arsys 2006-07-10 12:39:19.000000000 +0400
>>+++ ./include/asm-sparc/unistd.h 2006-08-10 17:08:19.000000000 +0400
>>@@ -318,6 +318,9 @@
>>#define __NR_unshare 299
>>#define __NR_set_robust_list 300
>>#define __NR_get_robust_list 301
>>+#define __NR_getluid 302
>>+#define __NR_setluid 303
>>+#define __NR_setublimit 304
>
>
> Hm, you seem to be ignoring this:
>
>
>>#ifdef __KERNEL__
>>/* WARNING: You MAY NOT add syscall numbers larger than 301, since
>
>
> Same thing for sparc64:
[...skipped...]

Oh, will fix NR_SYSCALLS in entry.S and the comment in unistd.h. Thanks for catching this!

Thanks,
Kirill

Subject: Re: [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by dev on Thu, 17 Aug 2006 12:03:43 GMT
View Forum Message <> Reply to Message

>>Add the following system calls for UB management:
>> 1. sys_getluid   - get current UB id
>> 2. sys_setluid   - changes exec_ and fork_ UBs on current
>> 3. sys_setublimit - set limits for resources consumtions
>>
>
>
> Why not have another system call for getting the current limits?
will add sys_getublimit().

> But as I said in previous mail, configfs seems like a better choice for
> user interface. That way user has to go to one place to read/write
> limits, see the current usage and other stats.
Check another email about interfaces. I have arguments against it :/

>>Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
>>Signed-Off-By: Kirill Korotaev <dev@sw.ru>
>
>
> ...<snip>...
>
>>+
>>+/*
>>+ * The setbeanlimit syscall
>>+ */
>>+asmlinkage long sys_setublimit(uid_t uid, unsigned long resource,
>>+ unsigned long *limits)
>>+{
>
>
>>+ ub = beancounter_findcreate(uid, NULL, 0);
>>+ if (ub == NULL)
>>+ goto out;
>>+
>>+ spin_lock_irqsave(&ub->ub_lock, flags);
>>+ ub->ub_parms[resource].barrier = new_limits[0];
>>+ ub->ub_parms[resource].limit = new_limits[1];

>>+ spin_unlock_irqrestore(&ub->ub_lock, flags);
>>+
>
>
> I think there should be a check here for seeing if the new limits are
> lower than the current usage of a resource. If so then take appropriate
> action.
any idea what exact action to add here?
Looks like can be added when needed, agree?

Thanks,
Kirill

Subject: Re: [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by dev on Thu, 17 Aug 2006 12:11:27 GMT
View Forum Message <> Reply to Message

Rohit Seth wrote:
> On Wed, 2006-08-16 at 20:04 +0100, Alan Cox wrote:
>
>>Ar Mer, 2006-08-16 am 11:17 -0700, ysgrifennodd Rohit Seth:
>>
>>>I think there should be a check here for seeing if the new limits are
>>>lower than the current usage of a resource. If so then take appropriate
>>>action.
>>
>>Generally speaking there isn't a sane appropriate action because the
>>resources can't just be yanked.
>>
>
>
> I was more thinking about (for example) user land physical memory limit
> for that bean counter. If the limits are going down, then the system
> call should try to flush out page cache pages or swap out anonymous
> memory. But you are right that it won't be possible in all cases, like
> for in kernel memory limits.
Such kind of memory management is less efficient than the one
making decisions based on global shortages and global LRU alogrithm.

The problem here is that doing swap out takes more expensive disk I/O
influencing other users.

So throttling algorithms if wanted should be optional, not mandatory.
Lets postpone it and concentrate on the core.

Thanks,
Kirill

Subject: Re: [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Greg KH on Thu, 17 Aug 2006 12:14:49 GMT
View Forum Message <> Reply to Message

On Thu, Aug 17, 2006 at 03:45:56PM +0400, Kirill Korotaev wrote:
> >>+struct user_beancounter
> >>+{
> >>+ atomic_t ub_refcount;
> > >
> > >
> > >Why not use a struct kref here instead of rolling your own reference
> > >counting logic?
> >
> > We need more complex decrement/locking scheme than krefs
> > provide. e.g. in __put_beancounter() we need
> > atomic_dec_and_lock_irqsave() semantics for performance optimizations.

Ah, ok, missed that. Nevermind then :)

thanks,

greg k-h

Subject: Re: Re: [ckrm-tech] [RFC][PATCH 3/7] UBC: ub context and inheritance
Posted by xemul on Thu, 17 Aug 2006 13:21:31 GMT
View Forum Message <> Reply to Message

Srivatsa Vaddagiri wrote:
> On Wed, Aug 16, 2006 at 07:38:44PM +0400, Kirill Korotaev wrote:
>
>> Contains code responsible for setting UB on task,
>> it's inheriting and setting host context in interrupts.
>>
>> Task references three beancounters:
>>   1. exec_ub current context. all resources are
>>   charged to this beancounter.
>>   2. task_ub beancounter to which task_struct is
>>   charged itself.
>>   3. fork_sub beancounter which is inherited by
>>   task's children on fork
>>
>
> Is there a case where exec_ub and fork_sub can differ? I dont see that
> in the patch.
>
Look in context changing in interrupts - "set_exec_ub(&ub0);" is done there.

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by dev on Thu, 17 Aug 2006 13:25:43 GMT
View Forum Message <> Reply to Message

> If I'm reading this patch right then seems like you are making page
> allocations to fail w/o (for example) trying to purge some pages from
> the page cache belonging to this container. Or is that reclaim going to
> come later?

charged kernel objects can't be _reclaimed_. how do you propose
to reclaim tasks page tables or files or task struct or vma or etc.?

Kirill

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by dev on Thu, 17 Aug 2006 13:29:43 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Wed, 2006-08-16 at 19:40 +0400, Kirill Korotaev wrote:
>
>>--- ./include/linux/mm.h.kmemcore	2006-08-16 19:10:38.000000000
>>+0400
>>+++ ./include/linux/mm.h	2006-08-16 19:10:51.000000000 +0400
>>@@ -274,8 +274,14 @@ struct page {
>>	unsigned int gfp_mask;
>>	unsigned long trace[8];
>> #endif
>>+#ifdef CONFIG_USER_RESOURCE
>>+	union {
>>+		struct user_beancounter *page_ub;
>>+	} bc;
>>+#endif
>> };
>
>
> Is everybody OK with adding this accounting to the 'struct page'? Is
> there any kind of noticeable performance penalty for this? I thought
> that we had this aligned pretty well on cacheline boundaries.
When I discussed this with Hugh Dickins on summit we agreed
that +4 bytes on page struct for kernel using accounting
are ok and almost unavoidable.

it can be stored not on the struct page, but in this
case you need to introduce some kind of hash to lookup ub
quickly from page, which is slower for accounting-enabled kernels.

> How many things actually use this? Can we have the slab ubcs without
> the struct page pointer?
slab doesn't use this pointer on the page.
It is used for pages allocated by buddy
alocator implicitly (e.g. LDT pages, page tables, ...).

Kirill

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by dev on Thu, 17 Aug 2006 13:33:44 GMT
View Forum Message <> Reply to Message

> On Wed, 2006-08-16 at 11:47 -0700, Dave Hansen wrote:
>
>>On Wed, 2006-08-16 at 19:40 +0400, Kirill Korotaev wrote:
>>
>>>--- ./include/linux/mm.h.kmemcore	2006-08-16 19:10:38.000000000
>>>+0400
>>>+++ ./include/linux/mm.h	2006-08-16 19:10:51.000000000 +0400
>>>@@ -274,8 +274,14 @@ struct page {
>>>	unsigned int gfp_mask;
>>>	unsigned long trace[8];
>>> #endif
>>>+#ifdef CONFIG_USER_RESOURCE
>>>+	union {
>>>+		struct user_beancounter *page_ub;
>>>+	} bc;
>>>+#endif
>>> };
>>
>>Is everybody OK with adding this accounting to the 'struct page'?
>
>
> My preference would be to have container (I keep on saying container,
> but resource beancounter) pointer embeded in task, mm(not sure),
> address_space and anon_vma structures. This should allow us to track
> user land pages optimally. But for tracking kernel usage on behalf of
> user, we will have to use an additional field (unless we can re-use
> mapping). Please correct me if I'm wrong, though all the kernel
> resources will be allocated/freed in context of a user process. And at
> that time we know if a allocation should succeed or not. So we may
> actually not need to track kernel pages that closely. We are not going
> to run reclaim on any of them anyways.
objects are really allocated in process context
(except for TCP/IP and other softirqs which are done in arbitrary
process context!)
And objects are not always freed in correct context (!).

Note, page_ub is not for _user_ pages. user pages accounting will be added
in next patch set. page_ub is added to track kernel allocations.

Kirill

Subject: Re: [RFC][PATCH 7/7] UBC: proc interface
Posted by dev on Thu, 17 Aug 2006 13:41:18 GMT
View Forum Message <> Reply to Message

> On Wed, Aug 16, 2006 at 07:44:30PM +0400, Kirill Korotaev wrote:
>
>>Add proc interface (/proc/user_beancounters) allowing to see current
>>state (usage/limits/fails for each UB). Implemented via seq files.
>
>
> Ugh, why /proc? This doesn't have anything to do with processes, just
> users, right? What's wrong with /sys/kernel/ instead?
We can move it, if there are much objections.
It is just here for more than 3 years (AFAIK starting from Alan's UBC)
and would be nice to have for compatibility (at least with existing OpenVZ).
But if it is required -- will do.

> Or /sys/kernel/debug/user_beancounters/ in debugfs as this is just a
> debugging thing, right?
debugfs is usually OFF imho. you don't export meminfo information in debugfs,
correct? user usages are the same imho...

Kirill

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by dev on Thu, 17 Aug 2006 13:45:00 GMT
View Forum Message <> Reply to Message

> The
>
> +	ub->ub_parms[UB_KMEMSIZE].limit = 32 * 1024 * 1024
>
> seems a bit arbitary. 32Mb is variously vast amounts of memory and not
> enough to boot depending if you are booting a PDA or a 4096 core Itanic
> box
this limit is for newly created UBs, host system (ub0) is
unlimited by default.
The idea was to limit the user by default to make system secure.
do you think it is good idea to have unlimited users created by default?

Anyway, after creating UB context normal behaviour would be to set
some limits.

Thanks,
Kirill

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by dev on Thu, 17 Aug 2006 13:53:34 GMT
View Forum Message <> Reply to Message

> On Wed, Aug 16, 2006 at 07:24:03PM +0400, Kirill Korotaev wrote:
>
>>As the first step we want to propose for discussion
>>the most complicated parts of resource management:
>>kernel memory and virtual memory.
>
> Do you have any plans to post a CPU controller? Is that tied to UBC
> interface as well?

Not everything at once :) To tell the truth I think CPU controller
is even more complicated than user memory accounting/limiting.

No, fair CPU scheduler is not tied to UBC in any regard.
As we discussed before, it is valuable to have an ability to limit
different resources separately (CPU, disk I/O, memory, etc.).
For example, it can be possible to place some mission critical
kernel threads (like kjournald) in a separate contanier.

This patches are related to kernel memory and nothing more :)

Thanks,
Kirill

Subject: Re: [ckrm-tech] [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by dev on Thu, 17 Aug 2006 14:00:36 GMT
View Forum Message <> Reply to Message

> On Wed, Aug 16, 2006 at 07:37:26PM +0400, Kirill Korotaev wrote:
>
>>+struct user_beancounter
>>+{
>>+ atomic_t ub_refcount;
>>+ spinlock_t ub_lock;
>>+ uid_t   ub_uid;
>>+ struct hlist_node hash;

>>+
>>+ struct user_beancounter *parent;
>
>
> This seems to hint at some heirarchy of ubc? How would that heirarchy be
> used? I cant find anything in the patch which forms this heirarchy
> (basically I dont see any place where beancounter_findcreate() is called
> with non-NULL 2nd arg).
yes, it is possible to use hierarchical beancounters.
kernel memory, user memory and TCP/IP buffers are accounted hierarchicaly.
user interface for this is not provided yet as it would complicate patchset
and increase number of topics for discussion :)

> [snip]
>
>
>>+static void init_beancounter_syslimits(struct user_beancounter *ub)
>>+{
>>+ int k;
>>+
>>+ for (k = 0; k < UB_RESOURCES; k++)
>>+ ub->ub_parms[k].barrier = ub->ub_parms[k].limit;
>
>
> This sets barrier to 0. Is this value of 0 interpreted differently by
> different controllers? One way to interpret it is "dont allocate any
> resource", other way to interpret it is "don't care - give me what you
> can" (which makes sense for stuff like CPU and network bandwidth).
every patch which adds a resource modifies this function and sets
some default limit. Check: [PATCH 5/7] UBC: kernel memory accounting (core)

Thanks,
Kirill

Subject: Re: [ckrm-tech] [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by dev on Thu, 17 Aug 2006 14:03:08 GMT

View Forum Message <> Reply to Message

Srivatsa Vaddagiri wrote:
> On Wed, Aug 16, 2006 at 07:39:43PM +0400, Kirill Korotaev wrote:
>
>
>>+/*
>>+ * The setbeanlimit syscall
>>+ */
>>+asmlinkage long sys_setublimit(uid_t uid, unsigned long resource,
>>+ unsigned long *limits)

>>+{
>
>
> [snip]
>
>
>>+ spin_lock_irqsave(&ub->ub_lock, flags);
>>+ ub->ub_parms[resource].barrier = new_limits[0];
>>+ ub->ub_parms[resource].limit = new_limits[1];
>
>
> Would it be usefull to notify the "resource" controller about this
> change in limits? For ex: in case of the CPU controller I wrote
> (http://lkml.org/lkml/2006/8/4/9), I was finding it usefull to recv
> notification of changes to these limits, so that internal structures
> (which are kept per-task-group) can be updated.
I think this can be added when needed, no?
See no much reason to add notifications which are not used yet.

Please, keep in mind. This patch set can be extended in infinite number
of ways. But!!! It contains only the required minimal functionality.
When we are to add code requiring to know about limit changes or fails
or whatever we can always extend it accordingly.

Thanks,
Kirill

Subject: Re: [ckrm-tech] [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Dave Hansen on Thu, 17 Aug 2006 14:32:40 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-17 at 15:45 +0400, Kirill Korotaev wrote:
> We need more complex decrement/locking scheme than krefs
> provide. e.g. in __put_beancounter() we need
> atomic_dec_and_lock_irqsave() semantics for performance optimizations.

Is it possible to put the locking in the destructor? It seems like that
should give similar behavior.

-- Dave

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Dave Hansen on Thu, 17 Aug 2006 14:36:19 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-17 at 17:31 +0400, Kirill Korotaev wrote:
> > How many things actually use this? Can we have the slab ubcs
> without
> > the struct page pointer?
> slab doesn't use this pointer on the page.
> It is used for pages allocated by buddy
> alocator implicitly (e.g. LDT pages, page tables, ...).

Hmmm. There aren't _that_ many of those cases, right? Are there any that absolutely need raw access to the buddy allocator? I'm pretty sure that pagetables can be moved over to a slab, as long as we bump up the alignment.

It does seem a wee bit silly to have the pointer in _all_ of the struct pages, even the ones for which we will never do any accounting (and even on kernels that never need it). But, a hashing scheme sounds like a fine idea.

-- Dave

Subject: Re: [ckrm-tech] [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Dave Hansen on Thu, 17 Aug 2006 14:38:13 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-17 at 17:27 +0400, Kirill Korotaev wrote:
> charged kernel objects can't be _reclaimed_. how do you propose
> to reclaim tasks page tables or files or task struct or vma or etc.?

Do you have any statistics on which of these objects are the most troublesome? If it _is_ pagetables, for instance, it is quite conceivable that we could reclaim them.

This one probably deserves a big, fat comment, though. ;)

-- Dave

Subject: Re: [RFC][PATCH 7/7] UBC: proc interface
Posted by Greg KH on Thu, 17 Aug 2006 15:40:23 GMT
View Forum Message <> Reply to Message

On Thu, Aug 17, 2006 at 05:43:16PM +0400, Kirill Korotaev wrote:
> >On Wed, Aug 16, 2006 at 07:44:30PM +0400, Kirill Korotaev wrote:
> >
> >>Add proc interface (/proc/user_beancounters) allowing to see current
> >>state (usage/limits/fails for each UB). Implemented via seq files.

> >
> >
> >Ugh, why /proc? This doesn't have anything to do with processes, just
> >users, right? What's wrong with /sys/kernel/ instead?
> We can move it, if there are much objections.

I am objecting. /proc is for processes so do not add any new files
there that do not deal with processes.

> >Or /sys/kernel/debug/user_beancounters/ in debugfs as this is just a
> >debugging thing, right?
> debugfs is usually OFF imho.

No, distros enable it.

> you don't export meminfo information in debugfs, correct?

That is because the meminfo is tied to processes, or was added to proc
before debugfs came about.

Then how about just /sys/kernel/ instead and use sysfs? Just remember,
one value per file please.

thanks,

greg k-h

Subject: Re: [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by Andrew Morton on Thu, 17 Aug 2006 15:40:33 GMT
View Forum Message <> Reply to Message

On Thu, 17 Aug 2006 16:13:30 +0400
Kirill Korotaev <dev@sw.ru> wrote:

> > I was more thinking about (for example) user land physical memory limit
> > for that bean counter. If the limits are going down, then the system
> > call should try to flush out page cache pages or swap out anonymous
> > memory. But you are right that it won't be possible in all cases, like
> > for in kernel memory limits.
> Such kind of memory management is less efficient than the one
> making decisions based on global shortages and global LRU alogrithm.

I also was quite surprised that openvz appears to have no way of
constraining a container's memory usage. "I want to run this bunch of
processes in a 4.5GB container".

> The problem here is that doing swap out takes more expensive disk I/O

> influencing other users.

A well-set-up container would presumably be working against its own
spindle(s). If the operator has gone to all the trouble of isolating a job
from the system's other jobs, he'd be pretty dumb to go and let all the
"isolated" jobs share a stinky-slow resource like a disk.

But yes, swap is a problem. To do this properly we'd need a way of saying
"this container here uses that swap device over there".

Subject: Re: Re: [RFC][PATCH 7/7] UBC: proc interface
Posted by kir on Thu, 17 Aug 2006 16:12:30 GMT
View Forum Message <> Reply to Message

Greg KH wrote:
> On Thu, Aug 17, 2006 at 05:43:16PM +0400, Kirill Korotaev wrote:
>
>>> On Wed, Aug 16, 2006 at 07:44:30PM +0400, Kirill Korotaev wrote:
>>>
>>>
>>>> Add proc interface (/proc/user_beancounters) allowing to see current
>>>> state (usage/limits/fails for each UB). Implemented via seq files.
>>>>
>>> Ugh, why /proc? This doesn't have anything to do with processes, just
>>> users, right? What's wrong with /sys/kernel/ instead?
>>>
>> We can move it, if there are much objections.
>>
>
> I am objecting. /proc is for processes so do not add any new files
> there that do not deal with processes.
>
>
>>> Or /sys/kernel/debug/user_beancounters/ in debugfs as this is just a
>>> debugging thing, right?
>>>
>> debugfs is usually OFF imho.
>>
>
> No, distros enable it.
>
>
>> you don't export meminfo information in debugfs, correct?
>>
>
> That is because the meminfo is tied to processes, or was added to proc
> before debugfs came about.

>
> Then how about just /sys/kernel/ instead and use sysfs? Just remember,
> one value per file please.
>
I see two problems with that. But let me first describe the current
/proc/user_beancounters. This is how it looks like from inside a container:

```
# cat /proc/user_beancounters
Version: 2.5
       uid  resource           held    maxheld    barrier      limit    failcnt
       123: kmemsize         836919    1005343    2752512    2936012          0
            lockedpages           0          0         32         32          0
            privvmpages        4587       7289      49152      53575          0
.....(more lines like that).....
```

I.e. a container owner can take a glance over the current parameters,
their usage and (the thing that is really important) fail counters. Fail
counter increases each time a parameter hits the limit. This is very
straightforward way for container's owner to see if everything is OK or not.

So, the problems with /sys are:

(1) Gettng such info from 40+ files requires at least some script, while
now cat is just fine.

(2) Do we want to virtualize sysfs and enable /sys for every container?
Note that user_beancounters statistics is really needed for container's
owner to see. At the same time, container's owner should not be able to
modify it -- so we should end up with read/write ubc entries for the
host system and read-only ones for the container.

Taking into account those two issues, current /proc/user_beancounters
might be not that bad.

Subject: Re: [ckrm-tech] [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by Srivatsa Vaddagiri on Thu, 17 Aug 2006 16:19:06 GMT
View Forum Message <> Reply to Message

On Thu, Aug 17, 2006 at 06:04:59PM +0400, Kirill Korotaev wrote:
> Please, keep in mind. This patch set can be extended in infinite number
> of ways. But!!! It contains only the required minimal functionality.

Sure ..But going by this it should mean that we don't see any code which
hints of heirarchy (->parent)? :)

> When we are to add code requiring to know about limit changes or fails

> or whatever we can always extend it accordingly.

--
Regards,
vatsa

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Rohit Seth on Thu, 17 Aug 2006 16:36:55 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-17 at 01:22 +0100, Alan Cox wrote:
> Ar Mer, 2006-08-16 am 12:15 -0700, ysgrifennodd Rohit Seth:
> > resources will be allocated/freed in context of a user process. And at
> > that time we know if a allocation should succeed or not. So we may
> > actually not need to track kernel pages that closely.
>
> Quite the reverse, tracking kernel pages is critical,

Having the knowledge of how many kernel pages are getting used by each container is indeed very useful. But as long as the context in which they are created and destroyed is identifiable, there is no need to really physically tag each page with container id. And for the cases where we have no context, it will be worth while to see if mapping field could be used.

> user pages kind of
> balance out for many cases kernel ones don't.
>

It is useful to put limits on some group of applications. So it is really not only about balancing out.

-rohit

Subject: Re: [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Rohit Seth on Thu, 17 Aug 2006 16:55:53 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-17 at 15:53 +0400, Kirill Korotaev wrote:
> Rohit Seth wrote:
> > On Wed, 2006-08-16 at 19:37 +0400, Kirill Korotaev wrote:
> >
> >>Core functionality and interfaces of UBC:
> >>find/create beancounter, initialization,

> >>charge/uncharge of resource, core objects' declarations.
> >>
> >>Basic structures:
> >> ubparm   - resource description
> >> user_beancounter - set of resources, id, lock
> >>
> >>Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
> >>Signed-Off-By: Kirill Korotaev <dev@sw.ru>
> >>
> >>---
> >> include/ub/beancounter.h | 157 ++++++++++++++++++
> >> init/main.c | 4
> >> kernel/Makefile | 1
> >> kernel/ub/Makefile | 7
> >> kernel/ub/beancounter.c | 398
++
> >> 5 files changed, 567 insertions(+)
> >>
> >>--- /dev/null 2006-07-18 14:52:43.075228448 +0400
> >>+++ ./include/ub/beancounter.h 2006-08-10 14:58:27.000000000 +0400
> >>@@ -0,0 +1,157 @@
> >>+/*
> >>+ * include/ub/beancounter.h
> >>+ *
> >>+ * Copyright (C) 2006 OpenVZ. SWsoft Inc
> >>+ *
> >>+ */
> >>+
> >>+#ifndef _LINUX_BEANCOUNTER_H
> >>+#define _LINUX_BEANCOUNTER_H
> >>+
> >>+/*
> >>+ * Resource list.
> >>+ */
> >>+
> >>+#define UB_RESOURCES 0
> >>+
> >>+struct ubparm {
> >>+ /*
> >>+ * 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 barrier;
> >>+ /* hard resource limit */
> >>+ unsigned long limit;
> >>+ /* consumed resources */
> >>+ unsigned long held;

> >>+ /* maximum amount of consumed resources through the last period */
> >>+ unsigned long maxheld;
> >>+ /* minimum amount of consumed resources through the last period */
> >>+ unsigned long minheld;
> >>+ /* count of failed charges */
> >>+ unsigned long failcnt;
> >>+};
> >
> >
> > What is the difference between barrier and limit. They both sound like
> > hard limits. No?
> check __charge_beancounter_locked and severity.
> It provides some kind of soft and hard limits.
>

Would be easier to just rename them as soft and hard limits...

> >>+
> >>+/*
> >>+ * Kernel internal part.
> >>+ */
> >>+
> >>+#ifdef __KERNEL__
> >>+
> >>+#include <linux/config.h>
> >>+#include <linux/spinlock.h>
> >>+#include <linux/list.h>
> >>+#include <asm/atomic.h>
> >>+
> >>+/*
> >>+ * UB_MAXVALUE is essentially LONG_MAX declared in a cross-compiling safe form.
> >>+ */
> >>+ /* resources statistics and settings */
> >>+ struct ubparm ub_parms[UB_RESOURCES];
> >>+};
> >>+
> >
> >
> > I presume UB_RESOURCES value is going to change as different resources
> > start getting tracked.
> what's wrong with it?
>

...just that user land will need to be some how informed about that.

> > I think something like configfs should be used for user interface. It
> > automatically presents the right interfaces to user land (based on
> > kernel implementation). And you wouldn't need any changes in glibc etc.

> 1. UBC doesn't require glibc modificatins.

You are right not for setting the limits. But for adding any new
functionality related to containers....as in you just added a new system
call to get the limits.

> 2. if you think a bit more about it, adding UB parameters doesn't
>   require user space changes as well.
> 3. it is possible to add any kind of interface for UBC. but do you like the idea
>   to grep 200(containers)x20(parameters) files for getting current usages?

How are you doing it currently and how much more efficient it is in
comparison to configfs?

>   Do you like the idea to convert numbers to strings and back w/o
>   thinking of data types?

IMO, setting up limits and containers (themselves) is not a common
operation.   I wouldn't be too worried about loosing those few extra
cycles in setting them up.

-rohit

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Rohit Seth on Thu, 17 Aug 2006 17:02:16 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-17 at 17:27 +0400, Kirill Korotaev wrote:
> > If I'm reading this patch right then seems like you are making page
> > allocations to fail w/o (for example) trying to purge some pages from
> > the page cache belonging to this container. Or is that reclaim going to
> > come later?
>
> charged kernel objects can't be _reclaimed_. how do you propose
> to reclaim tasks page tables or files or task struct or vma or etc.?

I agree that kernel objects cann't be reclaimed easily. But what you
are proposing is also not right. Returning failure w/o doing any
reclaim on pages (that are reclaimable) is not useful. And this is why
I asked, is this change going to be part of next set of patches (as
current set of patches are only tracking kernel usage).

-rohit

Subject: Re: [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by Rohit Seth on Thu, 17 Aug 2006 17:05:19 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-17 at 16:04 +0400, Kirill Korotaev wrote:
> >>Add the following system calls for UB management:
> >> 1. sys_getluid   - get current UB id
> >> 2. sys_setluid   - changes exec_ and fork_ UBs on current
> >> 3. sys_setublimit - set limits for resources consumtions
> >>
> >
> >
> > Why not have another system call for getting the current limits?
> will add sys_getublimit().
>
> > But as I said in previous mail, configfs seems like a better choice for
> > user interface. That way user has to go to one place to read/write
> > limits, see the current usage and other stats.
> Check another email about interfaces. I have arguments against it :/
>

...and I'm still not convinced that syscall is the right approach.

> > I think there should be a check here for seeing if the new limits are
> > lower than the current usage of a resource. If so then take appropriate
> > action.
> any idea what exact action to add here?
> Looks like can be added when needed, agree?
>

When you have the support of user memory, then operations like flush the
extra pages belonging to the container to disk seems reasonable.

-rohit

Subject: Re: [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by Rohit Seth on Thu, 17 Aug 2006 17:08:19 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-17 at 16:13 +0400, Kirill Korotaev wrote:
> Rohit Seth wrote:
> > On Wed, 2006-08-16 at 20:04 +0100, Alan Cox wrote:
> >
> >>Ar Mer, 2006-08-16 am 11:17 -0700, ysgrifennodd Rohit Seth:
> >>
> >>>I think there should be a check here for seeing if the new limits are
> >>>lower than the current usage of a resource. If so then take appropriate

> >>>action.
> >>
> >>Generally speaking there isn't a sane appropriate action because the
> >>resources can't just be yanked.
> >>
> >
> >
> > I was more thinking about (for example) user land physical memory limit
> > for that bean counter. If the limits are going down, then the system
> > call should try to flush out page cache pages or swap out anonymous
> > memory. But you are right that it won't be possible in all cases, like
> > for in kernel memory limits.
> Such kind of memory management is less efficient than the one
> making decisions based on global shortages and global LRU alogrithm.
>
> The problem here is that doing swap out takes more expensive disk I/O
> influencing other users.
>
> So throttling algorithms if wanted should be optional, not mandatory.
> Lets postpone it and concentrate on the core.
>

I'm really interested in seeing what changes you make in alloc_page when
the container limits are hit.

When a container is throttling then yes it will have some additional
cost to other containers but that is the cost of sharing an underlying
platform.

-rohit

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Rohit Seth on Thu, 17 Aug 2006 17:13:11 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-17 at 17:35 +0400, Kirill Korotaev wrote:

> > My preference would be to have container (I keep on saying container,
> > but resource beancounter) pointer embeded in task, mm(not sure),
> > address_space and anon_vma structures. This should allow us to track
> > user land pages optimally. But for tracking kernel usage on behalf of
> > user, we will have to use an additional field (unless we can re-use
> > mapping). Please correct me if I'm wrong, though all the kernel
> > resources will be allocated/freed in context of a user process. And at
> > that time we know if a allocation should succeed or not. So we may
> > actually not need to track kernel pages that closely. We are not going

> > to run reclaim on any of them anyways.
> objects are really allocated in process context
> (except for TCP/IP and other softirqs which are done in arbitrary
> process context!)

Can these pages be tagged using mapping field of the page struct.

> And objects are not always freed in correct context (!).
>
You mean beyond Networking and softirq.

> Note, page_ub is not for _user_ pages. user pages accounting will be added
> in next patch set. page_ub is added to track kernel allocations.
>

But will the page_ub be used for some purpose for user land pages?

-rohit

Subject: Re: [ckrm-tech] [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Chandra Seetharaman on Thu, 17 Aug 2006 18:59:15 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-17 at 18:02 +0400, Kirill Korotaev wrote:
<snip>
> >>+static void init_beancounter_syslimits(struct user_beancounter *ub)
> >>+{
> >>+ int k;
> >>+
> >>+ for (k = 0; k < UB_RESOURCES; k++)
> >>+ ub->ub_parms[k].barrier = ub->ub_parms[k].limit;
> >
> >
> > This sets barrier to 0. Is this value of 0 interpreted differently by
> > different controllers? One way to interpret it is "dont allocate any
> > resource", other way to interpret it is "don't care - give me what you
> > can" (which makes sense for stuff like CPU and network bandwidth).
> every patch which adds a resource modifies this function and sets
> some default limit. Check: [PATCH 5/7] UBC: kernel memory accounting (core)

The idea of upper layer code changing the lower layer's code doesn't
sound good. May be you can think of defining some interface to do it.

>
> Thanks,
> Kirill
>

>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Chandra Seetharaman on Thu, 17 Aug 2006 19:55:43 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-17 at 17:55 +0400, Kirill Korotaev wrote:
> > On Wed, Aug 16, 2006 at 07:24:03PM +0400, Kirill Korotaev wrote:
> >
> >>As the first step we want to propose for discussion
> >>the most complicated parts of resource management:
> >>kernel memory and virtual memory.
> >
> > Do you have any plans to post a CPU controller? Is that tied to UBC
> > interface as well?
>
> Not everything at once :) To tell the truth I think CPU controller
> is even more complicated than user memory accounting/limiting.
>
> No, fair CPU scheduler is not tied to UBC in any regard.

Not having the CPU controller on UBC doesn't sound good for the
infrastructure. IMHO, the infrastructure (for resource management) we
are going to have should be able to support different resource
controllers, without each controllers needing to have their own
infrastructure/interface etc.,

> As we discussed before, it is valuable to have an ability to limit
> different resources separately (CPU, disk I/O, memory, etc.).

Having ability to limit/control different resources separately not
necessarily mean we should have different infrastructure for each.

> For example, it can be possible to place some mission critical
> kernel threads (like kjournald) in a separate contanier.

I don't understand the comment above (in this context).
>
> This patches are related to kernel memory and nothing more :)
>
> Thanks,
> Kirill
>
>
> ---
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Matt Helsley on Fri, 18 Aug 2006 01:58:48 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-16 at 19:37 +0400, Kirill Korotaev wrote:
> Core functionality and interfaces of UBC:
> find/create beancounter, initialization,
> charge/uncharge of resource, core objects' declarations.
>
> Basic structures:
> ubparm - resource description
> user_beancounter - set of resources, id, lock
>
> Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
> Signed-Off-By: Kirill Korotaev <dev@sw.ru>
>
> ---
> include/ub/beancounter.h | 157 ++++++++++++++++++
> init/main.c | 4
> kernel/Makefile | 1
> kernel/ub/Makefile | 7

> kernel/ub/beancounter.c | 398 ++
> 5 files changed, 567 insertions(+)
>
> --- /dev/null 2006-07-18 14:52:43.075228448 +0400
> +++ ./include/ub/beancounter.h 2006-08-10 14:58:27.000000000 +0400
> @@ -0,0 +1,157 @@
> +/*
> + * include/ub/beancounter.h
> + *
> + * Copyright (C) 2006 OpenVZ. SWsoft Inc
> + *
> + */
> +
> +#ifndef _LINUX_BEANCOUNTER_H
> +#define _LINUX_BEANCOUNTER_H
> +
> +/*
> + * Resource list.
> + */
> +
> +#define UB_RESOURCES 0
> +
> +struct ubparm {
> + /*
> + * 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 barrier;
> + /* hard resource limit */
> + unsigned long limit;
> + /* consumed resources */
> + unsigned long held;
> + /* maximum amount of consumed resources through the last period */
> + unsigned long maxheld;
> + /* minimum amount of consumed resources through the last period */
> + unsigned long minheld;
> + /* count of failed charges */
> + unsigned long failcnt;
> +};
> +
> +/*
> + * Kernel internal part.
> + */
> +
> +#ifdef __KERNEL__
> +
> +#include <linux/config.h>

> +#include <linux/spinlock.h>
> +#include <linux/list.h>
> +#include <asm/atomic.h>
> +
> +/*
> + * UB_MAXVALUE is essentially LONG_MAX declared in a cross-compiling safe form.
> + */
> +#define UB_MAXVALUE ((1UL << (sizeof(unsigned long)*8-1)) - 1)
> +
> +
> +/*
> + * Resource management structures
> + * Serialization issues:
> + *   beancounter list management is protected via ub_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-ub spinlock
> + */
> +
> +struct user_beancounter
> +{
> + atomic_t ub_refcount;
> + spinlock_t ub_lock;
> + uid_t   ub_uid;
> + struct hlist_node hash;
> +
> + struct user_beancounter *parent;
> + void   *private_data;
> +
> + /* resources statistics and settings */
> + struct ubparm ub_parms[UB_RESOURCES];
> +};
> +
> +enum severity { UB_BARRIER, UB_LIMIT, UB_FORCE };
> +
> +/* Flags passed to beancounter_findcreate() */
> +#define UB_LOOKUP_SUB 0x01 /* Lookup subbeancounter */
> +#define UB_ALLOC 0x02 /* May allocate new one */
> +#define UB_ALLOC_ATOMIC 0x04 /* Allocate with GFP_ATOMIC */
> +
> +#define UB_HASH_SIZE 256
> +
> +#ifdef CONFIG_USER_RESOURCE
> +extern struct hlist_head ub_hash[];
> +extern spinlock_t ub_hash_lock;
> +
> +static inline void ub_adjust_held_minmax(struct user_beancounter *ub,
> + int resource)

```
> +{
> + if (ub->ub_parms[resource].maxheld < ub->ub_parms[resource].held)
> +  ub->ub_parms[resource].maxheld = ub->ub_parms[resource].held;
> + if (ub->ub_parms[resource].minheld > ub->ub_parms[resource].held)
> +  ub->ub_parms[resource].minheld = ub->ub_parms[resource].held;
> +}
> +
> +void ub_print_resource_warning(struct user_beancounter *ub, int res,
> +  char *str, unsigned long val, unsigned long held);
> +void ub_print_uid(struct user_beancounter *ub, char *str, int size);
> +
> +int  __charge_beancounter_locked(struct user_beancounter *ub,
> +  int resource, unsigned long val, enum severity strict);
> +void charge_beancounter_notop(struct user_beancounter *ub,
> +  int resource, unsigned long val);
> +int  charge_beancounter(struct user_beancounter *ub,
> +  int resource, unsigned long val, enum severity strict);
> +
> +void __uncharge_beancounter_locked(struct user_beancounter *ub,
> +  int resource, unsigned long val);
> +void uncharge_beancounter_notop(struct user_beancounter *ub,
> +  int resource, unsigned long val);
> +void uncharge_beancounter(struct user_beancounter *ub,
> +  int resource, unsigned long val);
> +
> +struct user_beancounter *beancounter_findcreate(uid_t uid,
> +  struct user_beancounter *parent, int flags);
> +
> +static inline struct user_beancounter *get_beancounter(
> +  struct user_beancounter *ub)
> +{
> + atomic_inc(&ub->ub_refcount);
> + return ub;
> +}
> +
> +void __put_beancounter(struct user_beancounter *ub);
> +static inline void put_beancounter(struct user_beancounter *ub)
> +{
> + __put_beancounter(ub);
> +}
> +
> +void ub_init_early(void);
> +void ub_init_late(void);
> +void ub_init_proc(void);
> +
> +extern struct user_beancounter ub0;
> +extern const char *ub_rnames[];
> +
```

> +#else /* CONFIG_USER_RESOURCE */
> +
> +#define beancounter_findcreate(id, p, f) (NULL)
> +#define get_beancounter(ub)   (NULL)
> +#define put_beancounter(ub)   do { } while (0)
> +#define __charge_beancounter_locked(ub, r, v, s) (0)
> +#define charge_beancounter(ub, r, v, s)   (0)
> +#define charge_beancounter_notop(ub, r, v) do { } while (0)
> +#define __uncharge_beancounter_locked(ub, r, v) do { } while (0)
> +#define uncharge_beancounter(ub, r, v)   do { } while (0)
> +#define uncharge_beancounter_notop(ub, r, v) do { } while (0)
> +#define ub_init_early()   do { } while (0)
> +#define ub_init_late()   do { } while (0)
> +#define ub_init_proc()   do { } while (0)
> +
> +#endif /* CONFIG_USER_RESOURCE */
> +#endif /* __KERNEL__ */
> +
> +#endif /* _LINUX_BEANCOUNTER_H */
> --- ./init/main.c.ubcore 2006-08-10 14:55:47.000000000 +0400
> +++ ./init/main.c 2006-08-10 14:57:01.000000000 +0400
> @@ -52,6 +52,8 @@
> #include <linux/debug_locks.h>
> #include <linux/lockdep.h>
>
> +#include <ub/beancounter.h>
> +
> #include <asm/io.h>
> #include <asm/bugs.h>
> #include <asm/setup.h>
> @@ -470,6 +472,7 @@ asmlinkage void __init start_kernel(void
>   early_boot_irqs_off();
>   early_init_irq_lock_class();
>
> + ub_init_early();
> /*
>   * Interrupts are still disabled. Do necessary setups, then
>   * enable them
> @@ -563,6 +566,7 @@ asmlinkage void __init start_kernel(void
> #endif
>   fork_init(num_physpages);
>   proc_caches_init();
> + ub_init_late();
>   buffer_init();
>   unnamed_dev_init();
>   key_init();
> --- ./kernel/Makefile.ubcore 2006-08-10 14:55:47.000000000 +0400
> +++ ./kernel/Makefile 2006-08-10 14:57:01.000000000 +0400

```
> @@ -12,6 +12,7 @@ obj-y     = sched.o fork.o exec_domain.o
>
>  obj-$(CONFIG_STACKTRACE) += stacktrace.o
>  obj-y += time/
> +obj-y += ub/
>  obj-$(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/ub/Makefile 2006-08-10 14:57:01.000000000 +0400
> @@ -0,0 +1,7 @@
> +#
> +# User resources part (UBC)
> +#
> +# Copyright (C) 2006 OpenVZ. SWsoft Inc
> +#
> +
> +obj-$(CONFIG_USER_RESOURCE) += beancounter.o
> --- /dev/null 2006-07-18 14:52:43.075228448 +0400
> +++ ./kernel/ub/beancounter.c 2006-08-10 15:09:34.000000000 +0400
> @@ -0,0 +1,398 @@
> +/*
> + *  kernel/ub/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 <ub/beancounter.h>
> +
> +static kmem_cache_t *ub_cachep;
> +static struct user_beancounter default_beancounter;
> +static struct user_beancounter default_subbeancounter;
> +
> +static void init_beancounter_struct(struct user_beancounter *ub, uid_t id);
> +
> +struct user_beancounter ub0;
> +
> +const char *ub_rnames[] = {
> +};
> +
> +#define ub_hash_fun(x) ((((x) >> 8) ^ (x)) & (UB_HASH_SIZE - 1))
> +#define ub_subhash_fun(p, id) ub_hash_fun((p)->ub_uid + (id) * 17)
> +
```

> +struct hlist_head ub_hash[UB_HASH_SIZE];
> +spinlock_t ub_hash_lock;
> +
> +EXPORT_SYMBOL(ub_hash);
> +EXPORT_SYMBOL(ub_hash_lock);
> +
> +/*
> + * Per user resource beancounting. Resources are tied to their luid.

You haven't explained what an luid is at this point in the patch series.
Patch 0 says:

diff-ubc-syscalls.patch:
    Patch adds system calls for UB management:
    1. sys_getluid   - get current UB id

But I have no idea what that l is there for. Why not sys_get_ubid() for
instance?

> + * 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.
> + */
> +

So we create one beancounter object for every resource the user's tasks
allocate? For instance, one per socket? Or does "resource structure"
refer to something else?

> +struct user_beancounter *beancounter_findcreate(uid_t uid,
> + struct user_beancounter *p, int mask)
> +{
> + struct user_beancounter *new_ub, *ub, *tmpl_ub;
> + unsigned long flags;
> + struct hlist_head *slot;
> + struct hlist_node *pos;
> +
> + if (mask & UB_LOOKUP_SUB) {
> + WARN_ON(p == NULL);
> + tmpl_ub = &default_subbeancounter;
> + slot = &ub_hash[ub_subhash_fun(p, uid)];
> + } else {
> + WARN_ON(p != NULL);

> + tmpl_ub = &default_beancounter;
> + slot = &ub_hash[ub_hash_fun(uid)];
> + }
> + new_ub = NULL;
> +
> +retry:
> + spin_lock_irqsave(&ub_hash_lock, flags);
> + hlist_for_each_entry (ub, pos, slot, hash)
> + if (ub->ub_uid == uid && ub->parent == p)
> + break;
> +
> + if (pos != NULL) {
> + get_beancounter(ub);
> + spin_unlock_irqrestore(&ub_hash_lock, flags);
> +
> + if (new_ub != NULL) {
> + put_beancounter(new_ub->parent);
> + kmem_cache_free(ub_cachep, new_ub);
> + }
> + return ub;
> + }
> +
> + if (!(mask & UB_ALLOC))
> + goto out_unlock;
> +
> + if (new_ub != NULL)
> + goto out_install;
> +
> + if (mask & UB_ALLOC_ATOMIC) {

This block..

> + new_ub = kmem_cache_alloc(ub_cachep, GFP_ATOMIC);
> + if (new_ub == NULL)
> + goto out_unlock;
> +
> + memcpy(new_ub, tmpl_ub, sizeof(*new_ub));
> + init_beancounter_struct(new_ub, uid);
> + if (p)
> + new_ub->parent = get_beancounter(p);

ending here is almost exactly the same as the block ..

> + goto out_install;
> + }
> +
> + spin_unlock_irqrestore(&ub_hash_lock, flags);
> +

>From here..

> + new_ub = kmem_cache_alloc(ub_cachep, GFP_KERNEL);
> + if (new_ub == NULL)
> + goto out;
> +
> + memcpy(new_ub, tmpl_ub, sizeof(*new_ub));
> + init_beancounter_struct(new_ub, uid);
> + if (p)
> + new_ub->parent = get_beancounter(p);

to here. You could make a flag variable that holds GFP_ATOMIC or
GFP_KERNEL based on mask & UB_ALLOC_ATOMIC and perhaps turn this block
into a small helper.

> + goto retry;
> +
> +out_install:
> + hlist_add_head(&new_ub->hash, slot);
> +out_unlock:
> + spin_unlock_irqrestore(&ub_hash_lock, flags);
> +out:
> + return new_ub;
> +}
> +
> +EXPORT_SYMBOL(beancounter_findcreate);
> +
> +void ub_print_uid(struct user_beancounter *ub, char *str, int size)
> +{
> + if (ub->parent != NULL)
> + snprintf(str, size, "%u.%u", ub->parent->ub_uid, ub->ub_uid);
> + else
> + snprintf(str, size, "%u", ub->ub_uid);
> +}
> +
> +EXPORT_SYMBOL(ub_print_uid);

>From what I can see this patch doesn't really justify the need for the
EXPORT_SYMBOL. Shouldn't that be done in the patch where it's needed
outside of the kernel/ub code itself?

> +void ub_print_resource_warning(struct user_beancounter *ub, int res,
> + char *str, unsigned long val, unsigned long held)
> +{
> + char uid[64];
> +
> + ub_print_uid(ub, uid, sizeof(uid));

```
> + printk(KERN_WARNING "UB %s %s warning: %s "
> +   "(held %lu, fails %lu, val %lu)\n",
> +   uid, ub_rnames[res], str,
> +   (res < UB_RESOURCES ? ub->ub_parms[res].held : held),
> +   (res < UB_RESOURCES ? ub->ub_parms[res].failcnt : 0),
> +   val);
> +}
> +
> +EXPORT_SYMBOL(ub_print_resource_warning);
> +
> +static inline void verify_held(struct user_beancounter *ub)
> +{
> + int i;
> +
> + for (i = 0; i < UB_RESOURCES; i++)
> +  if (ub->ub_parms[i].held != 0)
> +   ub_print_resource_warning(ub, i,
> +     "resource is held on put", 0, 0);
> +}
> +
> +void __put_beancounter(struct user_beancounter *ub)
> +{
> + unsigned long flags;
> + struct user_beancounter *parent;
> +
> +again:
> + parent = ub->parent;
> + /* equevalent to atomic_dec_and_lock_irqsave() */
```

nit: s/que/qui/

```
> + local_irq_save(flags);
> + if (likely(!atomic_dec_and_lock(&ub->ub_refcount, &ub_hash_lock))) {
> +  if (unlikely(atomic_read(&ub->ub_refcount) < 0))
> +   printk(KERN_ERR "UB: Bad ub refcount: ub=%p, "
> +     "luid=%d, ref=%d\n",
> +     ub, ub->ub_uid,
> +     atomic_read(&ub->ub_refcount));
```

This seems to be for debugging purposes only. If not, perhaps this
printk ought to be rate limited?

```
> +  local_irq_restore(flags);
> +  return;
> + }
> +
> + if (unlikely(ub == &ub0)) {
> +  printk(KERN_ERR "Trying to put ub0\n");
```

Same thing for this printk.

```
> +  spin_unlock_irqrestore(&ub_hash_lock, flags);
> +  return;
> + }
> +
> + verify_held(ub);
> + hlist_del(&ub->hash);
> + spin_unlock_irqrestore(&ub_hash_lock, flags);
> +
> + kmem_cache_free(ub_cachep, ub);
> +
> + ub = parent;
> + if (ub != NULL)
> +  goto again;
```

Couldn't this be replaced by a do { } while (ub != NULL); loop?

```
> +}
> +
> +EXPORT_SYMBOL(__put_beancounter);
> +
> +/*
> + * Generic resource charging stuff
> + */
> +
> +int __charge_beancounter_locked(struct user_beancounter *ub,
> +  int resource, unsigned long val, enum severity strict)
> +{
> + /*
> +  * ub_value <= UB_MAXVALUE, value <= UB_MAXVALUE, and only one addition
> +  * at the moment is possible so an overflow is impossible.
> +  */
> + ub->ub_parms[resource].held += val;
> +
> + switch (strict) {
> +  case UB_BARRIER:
> +   if (ub->ub_parms[resource].held >
> +     ub->ub_parms[resource].barrier)
> +    break;
> +   /* fallthrough */
> +  case UB_LIMIT:
> +   if (ub->ub_parms[resource].held >
> +     ub->ub_parms[resource].limit)
> +    break;
> +   /* fallthrough */
> +  case UB_FORCE:
```

> +   ub_adjust_held_minmax(ub, resource);
> +   return 0;
> + default:
> +   BUG();
> + }
> +
> + ub->ub_parms[resource].failcnt++;
> + ub->ub_parms[resource].held -= val;
> + return -ENOMEM;
> +}
> +
> +int charge_beancounter(struct user_beancounter *ub,
> + int resource, unsigned long val, enum severity strict)
> +{
> + int retval;
> + struct user_beancounter *p, *q;
> + unsigned long flags;
> +
> + retval = -EINVAL;
> + BUG_ON(val > UB_MAXVALUE);
> +
> + local_irq_save(flags);

<factor>

> + for (p = ub; p != NULL; p = p->parent) {

Seems rather expensive to walk up the tree for every charge. Especially if the administrator wants a fine degree of resource control and makes a tall tree. This would be a problem especially when it comes to resources that require frequent and fast allocation.

> + spin_lock(&p->ub_lock);
> + retval = __charge_beancounter_locked(p, resource, val, strict);
> + spin_unlock(&p->ub_lock);
> + if (retval)
> +   goto unroll;

This can be factored by passing a flag that breaks the loop on an error:

```
  if (retval && do_break_err)
   return retval;
```

> + }

</factor>

> +out_restore:

> + local_irq_restore(flags);
> + return retval;
> +

<factor>

> +unroll:
> + for (q = ub; q != p; q = q->parent) {
> + spin_lock(&q->ub_lock);
> + __uncharge_beancounter_locked(q, resource, val);
> + spin_unlock(&q->ub_lock);
> + }

</factor>

> + goto out_restore;
> +}
> +
> +EXPORT_SYMBOL(charge_beancounter);
> +
> +void charge_beancounter_notop(struct user_beancounter *ub,
> + int resource, unsigned long val)
> +{
> + struct user_beancounter *p;
> + unsigned long flags;
> +
> + local_irq_save(flags);

<factor>

> + for (p = ub; p->parent != NULL; p = p->parent) {
> + spin_lock(&p->ub_lock);
> + __charge_beancounter_locked(p, resource, val, UB_FORCE);
> + spin_unlock(&p->ub_lock);
> + }

<factor>

> + local_irq_restore(flags);

Again, this could be factored with charge_beancounter using a helper
function.

> +}
> +
> +EXPORT_SYMBOL(charge_beancounter_notop);
> +
> +void __uncharge_beancounter_locked(struct user_beancounter *ub,

```
> +  int resource, unsigned long val)
> +{
> + if (unlikely(ub->ub_parms[resource].held < val)) {
> +  ub_print_resource_warning(ub, resource,
> +    "uncharging too much", val, 0);
> +  val = ub->ub_parms[resource].held;
> + }
> + ub->ub_parms[resource].held -= val;
> + ub_adjust_held_minmax(ub, resource);
> +}
> +
> +void uncharge_beancounter(struct user_beancounter *ub,
> +  int resource, unsigned long val)
> +{
> + unsigned long flags;
> + struct user_beancounter *p;
> +
> + for (p = ub; p != NULL; p = p->parent) {
> +  spin_lock_irqsave(&p->ub_lock, flags);
> +  __uncharge_beancounter_locked(p, resource, val);
> +  spin_unlock_irqrestore(&p->ub_lock, flags);
> + }
> +}
```

Looks like your unroll: label in charge_beancounter above.

```
> +
> +EXPORT_SYMBOL(uncharge_beancounter);
> +
> +void uncharge_beancounter_notop(struct user_beancounter *ub,
> +  int resource, unsigned long val)
> +{
> + struct user_beancounter *p;
> + unsigned long flags;
> +
> + local_irq_save(flags);
```

<factor>

```
> + for (p = ub; p->parent != NULL; p = p->parent) {
> +  spin_lock(&p->ub_lock);
> +  __uncharge_beancounter_locked(p, resource, val);
> +  spin_unlock(&p->ub_lock);
> + }
```

</factor>

```
> + local_irq_restore(flags);
```

```
> +}
> +
> +EXPORT_SYMBOL(uncharge_beancounter_notop);
> +
> +/*
> + * Initialization
> + *
> + * struct user_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 user_beancounter *ub, uid_t id)
> +{
> + atomic_set(&ub->ub_refcount, 1);
> + spin_lock_init(&ub->ub_lock);
> + ub->ub_uid = id;
> +}
> +
> +static void init_beancounter_nolimits(struct user_beancounter *ub)
> +{
> + int k;
> +
> + for (k = 0; k < UB_RESOURCES; k++) {
> +  ub->ub_parms[k].limit = UB_MAXVALUE;
> +  ub->ub_parms[k].barrier = UB_MAXVALUE;
> + }
> +}
> +
> +static void init_beancounter_syslimits(struct user_beancounter *ub)
> +{
> + int k;
> +
> + for (k = 0; k < UB_RESOURCES; k++)
> +  ub->ub_parms[k].barrier = ub->ub_parms[k].limit;
> +}
> +
> +void __init ub_init_early(void)
> +{
> + struct user_beancounter *ub;
> + struct hlist_head *slot;
> +
> + ub = &ub0;
> +
```

<factor>

> + memset(ub, 0, sizeof(*ub));
> + init_beancounter_nolimits(ub);
> + init_beancounter_struct(ub, 0);
> +

</factor>

> + spin_lock_init(&ub_hash_lock);
> + slot = &ub_hash[ub_hash_fun(ub->ub_uid)];
> + hlist_add_head(&ub->hash, slot);
> +}
> +
> +void __init ub_init_late(void)
> +{
> + struct user_beancounter *ub;
> +
> + ub_cachep = kmem_cache_create("user_beancounters",
> +   sizeof(struct user_beancounter),
> +   0, SLAB_HWCACHE_ALIGN, NULL, NULL);
> + if (ub_cachep == NULL)
> + panic("Can't create ubc caches\n");
> +
> + ub = &default_beancounter;

<factor>

> + memset(ub, 0, sizeof(default_beancounter));
> + init_beancounter_syslimits(ub);
> + init_beancounter_struct(ub, 0);
> +

</factor>

> + ub = &default_subbeancounter;

<factor>

> + memset(ub, 0, sizeof(default_subbeancounter));
> + init_beancounter_nolimits(ub);
> + init_beancounter_struct(ub, 0);

</factor>

> +}

Cheers,
 -Matt Helsley

Subject: Re: [ckrm-tech] [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by Matt Helsley on Fri, 18 Aug 2006 02:31:40 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-16 at 19:39 +0400, Kirill Korotaev wrote:
> Add the following system calls for UB management:
> 1. sys_getluid   - get current UB id
> 2. sys_setluid   - changes exec_ and fork_ UBs on current
> 3. sys_setublimit - set limits for resources consumtions
>
> Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
> Signed-Off-By: Kirill Korotaev <dev@sw.ru>
>
> ---
> arch/i386/kernel/syscall_table.S | 3
> arch/ia64/kernel/entry.S | 3
> arch/sparc/kernel/systbls.S | 2
> arch/sparc64/kernel/systbls.S | 2
> include/asm-i386/unistd.h | 5 +
> include/asm-ia64/unistd.h | 5 +
> include/asm-powerpc/systbl.h | 3
> include/asm-powerpc/unistd.h | 5 +
> include/asm-sparc/unistd.h | 3
> include/asm-sparc64/unistd.h | 3
> include/asm-x86_64/unistd.h | 8 ++
> kernel/ub/Makefile | 1
> kernel/ub/sys.c | 126 ++
> 13 files changed, 163 insertions(+), 6 deletions(-)
>
> --- ./arch/i386/kernel/syscall_table.S.ubsys 2006-07-10 12:39:10.000000000 +0400
> +++ ./arch/i386/kernel/syscall_table.S 2006-07-31 14:36:59.000000000 +0400
> @@ -317,3 +317,6 @@ ENTRY(sys_call_table)
> .long sys_vmsplice
> .long sys_move_pages
> .long sys_getcpu
> + .long sys_getluid
> + .long sys_setluid
> + .long sys_setublimit /* 320 */
> --- ./arch/ia64/kernel/entry.S.ubsys 2006-07-10 12:39:10.000000000 +0400
> +++ ./arch/ia64/kernel/entry.S 2006-07-31 15:25:36.000000000 +0400
> @@ -1610,5 +1610,8 @@ sys_call_table:
> data8 sys_sync_file_range // 1300
> data8 sys_tee
> data8 sys_vmsplice

> + daat8 sys_getluid
> + data8 sys_setluid
> + data8 sys_setublimit   // 1305
>
>   .org sys_call_table + 8*NR_syscalls // guard against failures to increase NR_syscalls
> --- ./arch/sparc/kernel/systbls.S.arsys 2006-07-10 12:39:10.000000000 +0400
> +++ ./arch/sparc/kernel/systbls.S 2006-08-10 17:07:15.000000000 +0400
> @@ -78,7 +78,7 @@ 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_getluid, sys_setluid, sys_setublimit
>
> #ifdef CONFIG_SUNOS_EMUL
>   /* Now the SunOS syscall table. */
> --- ./arch/sparc64/kernel/systbls.S.arsys 2006-07-10 12:39:11.000000000 +0400
> +++ ./arch/sparc64/kernel/systbls.S 2006-08-10 17:08:52.000000000 +0400
> @@ -79,7 +79,7 @@ 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_getluid,
sys_setluid, sys_setublimit
>
> #endif /* CONFIG_COMPAT */
>
> --- ./include/asm-i386/unistd.h.ubsys 2006-07-10 12:39:19.000000000 +0400
> +++ ./include/asm-i386/unistd.h 2006-07-31 15:56:31.000000000 +0400
> @@ -323,10 +323,13 @@
> #define __NR_vmsplice 316
> #define __NR_move_pages 317
> #define __NR_getcpu 318
> +#define __NR_getluid 319
> +#define __NR_setluid 320
> +#define __NR_setublimit 321
>
> #ifdef __KERNEL__
>
> -#define NR_syscalls 318
> +#define NR_syscalls 322
> #include <linux/err.h>
>
> /*
> --- ./include/asm-ia64/unistd.h.ubsys 2006-07-10 12:39:19.000000000 +0400
> +++ ./include/asm-ia64/unistd.h 2006-07-31 15:57:23.000000000 +0400
> @@ -291,11 +291,14 @@

> #define __NR_sync_file_range 1300
> #define __NR_tee   1301
> #define __NR_vmsplice   1302
> +#define __NR_getluid   1303
> +#define __NR_setluid   1304
> +#define __NR_setublimit   1305
>
> #ifdef __KERNEL__
>
>
> -#define NR_syscalls   279 /* length of syscall table */
> +#define NR_syscalls   282 /* length of syscall table */
>
> #define __ARCH_WANT_SYS_RT_SIGACTION
>
> --- ./include/asm-powerpc/systbl.h.arsys 2006-07-10 12:39:19.000000000 +0400
> +++ ./include/asm-powerpc/systbl.h 2006-08-10 17:05:53.000000000 +0400
> @@ -304,3 +304,6 @@ SYSCALL_SPU(fchmodat)
> SYSCALL_SPU(faccessat)
> COMPAT_SYS_SPU(get_robust_list)
> COMPAT_SYS_SPU(set_robust_list)
> +SYSCALL(sys_getluid)
> +SYSCALL(sys_setluid)
> +SYSCALL(sys_setublimit)
> --- ./include/asm-powerpc/unistd.h.arsys 2006-07-10 12:39:19.000000000 +0400
> +++ ./include/asm-powerpc/unistd.h 2006-08-10 17:06:28.000000000 +0400
> @@ -323,10 +323,13 @@
> #define __NR_faccessat 298
> #define __NR_get_robust_list 299
> #define __NR_set_robust_list 300
> +#define __NR_getluid 301
> +#define __NR_setluid 302
> +#define __NR_setublimit 303
>
> #ifdef __KERNEL__
>
> -#define __NR_syscalls 301
> +#define __NR_syscalls 304
>
> #define __NR__exit __NR_exit
> #define NR_syscalls __NR_syscalls
> --- ./include/asm-sparc/unistd.h.arsys 2006-07-10 12:39:19.000000000 +0400
> +++ ./include/asm-sparc/unistd.h 2006-08-10 17:08:19.000000000 +0400
> @@ -318,6 +318,9 @@
> #define __NR_unshare 299
> #define __NR_set_robust_list 300
> #define __NR_get_robust_list 301
> +#define __NR_getluid 302

> +#define __NR_setluid 303
> +#define __NR_setublimit 304
>
> #ifdef __KERNEL__
> /* WARNING: You MAY NOT add syscall numbers larger than 301, since
> --- ./include/asm-sparc64/unistd.h.arsys 2006-07-10 12:39:19.000000000 +0400
> +++ ./include/asm-sparc64/unistd.h 2006-08-10 17:09:24.000000000 +0400
> @@ -320,6 +320,9 @@
> #define __NR_unshare 299
> #define __NR_set_robust_list 300
> #define __NR_get_robust_list 301
> +#define __NR_getluid 302
> +#define __NR_setluid 303
> +#define __NR_setublimit 304
>
> #ifdef __KERNEL__
> /* WARNING: You MAY NOT add syscall numbers larger than 301, since
> --- ./include/asm-x86_64/unistd.h.ubsys 2006-07-10 12:39:19.000000000 +0400
> +++ ./include/asm-x86_64/unistd.h 2006-07-31 16:00:01.000000000 +0400
> @@ -619,10 +619,16 @@ __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_getluid 280
> +__SYSCALL(__NR_getluid, sys_getluid)
> +#define __NR_setluid 281
> +__SYSCALL(__NR_setluid, sys_setluid)
> +#define __NR_setublimit 282
> +__SYSCALL(__NR_setublimit, sys_setublimit)
>
> #ifdef __KERNEL__
>
> -#define __NR_syscall_max __NR_move_pages
> +#define __NR_syscall_max __NR_setublimit
> #include <linux/err.h>
>
> #ifndef __NO_STUBS
> --- ./kernel/ub/Makefile.ubsys 2006-07-28 14:08:37.000000000 +0400
> +++ ./kernel/ub/Makefile 2006-08-01 11:08:39.000000000 +0400
> @@ -6,3 +6,4 @@
>
> obj-$(CONFIG_USER_RESOURCE) += beancounter.o
> obj-$(CONFIG_USER_RESOURCE) += misc.o
> +obj-y += sys.o
> --- ./kernel/ub/sys.c.ubsys 2006-07-28 18:52:18.000000000 +0400
> +++ ./kernel/ub/sys.c 2006-08-03 16:14:23.000000000 +0400
> @@ -0,0 +1,126 @@
> +/*

> + * kernel/ub/sys.c
> + *
> + * Copyright (C) 2006 OpenVZ. SWsoft Inc
> + *
> + */
> +
> +#include <linux/config.h>
> +#include <linux/sched.h>
> +#include <asm/uaccess.h>
> +
> +#include <ub/beancounter.h>
> +#include <ub/task.h>
> +
> +#ifndef CONFIG_USER_RESOURCE

Get rid of the #ifdef since this file should only be compiled if
CONFIG_USER_RESOURCE=y anyway.

> +asmlinkage long sys_getluid(void)
> +{
> + return -ENOSYS;
> +}
> +
> +asmlinkage long sys_setluid(uid_t uid)
> +{
> + return -ENOSYS;
> +}
> +
> +asmlinkage long sys_setublimit(uid_t uid, unsigned long resource,
> + unsigned long *limits)
> +{
> + return -ENOSYS;
> +}

Looks to me like you want to add:

cond_syscall(sys_getluid);
...

in kernel/sys_ni.c and then you won't have to worry about making these
empty functions.

> +#else /* CONFIG_USER_RESOURCE */
> +
> +/*
> + * The (rather boring) getluid syscall
> + */
> +asmlinkage long sys_getluid(void)

```
> +{
> + struct user_beancounter *ub;
> +
> + ub = get_exec_ub();
> + if (ub == NULL)
> +  return -EINVAL;
> +
> + return ub->ub_uid;
> +}
> +
> +/*
> + * The setluid syscall
> + */
> +asmlinkage long sys_setluid(uid_t uid)
> +{
> + int error;
> + struct user_beancounter *ub;
> + struct task_beancounter *task_bc;
> +
> + task_bc = &current->task_bc;
> +
> + /* You may not disown a setluid */
> + error = -EINVAL;
> + if (uid == (uid_t)-1)
> +  goto out;
> +
> + /* You may only set an ub as root */
> + error = -EPERM;
> + if (!capable(CAP_SETUID))
> +  goto out;
```

With resource groups you don't necessarily have to be root -- just the owner of the group and task.

Filesystems and appropriate share representations offer a way to give regular users the ability to manage their resources without requiring CAP_FOO.

```
> + /* Ok - set up a beancounter entry for this user */
> + error = -ENOBUFS;
> + ub = beancounter_findcreate(uid, NULL, UB_ALLOC);
> + if (ub == NULL)
> +  goto out;
> +
> + /* install bc */
> + put_beancounter(task_bc->exec_ub);
> + task_bc->exec_ub = ub;
> + put_beancounter(task_bc->fork_sub);
```

> + task_bc->fork_sub = get_beancounter(ub);
> + error = 0;
> +out:
> + return error;
> +}
> +
> +/*
> + * The setbeanlimit syscall
> + */
> +asmlinkage long sys_setublimit(uid_t uid, unsigned long resource,
> + unsigned long *limits)
> +{
> + int error;
> + unsigned long flags;
> + struct user_beancounter *ub;
> + unsigned long new_limits[2];
> +
> + error = -EPERM;
> + if(!capable(CAP_SYS_RESOURCE))
> + goto out;

Again, a filesystem interface would give us more flexibility when it
comes to allowing users to manage their resources while still preventing
them from exceeding limits.

I doubt you really want to give owners of a container CAP_SYS_RESOURCE
and CAP_USER (i.e. total control over resource management) just to allow
them to manage their subset of the resources.

> + error = -EINVAL;
> + if (resource >= UB_RESOURCES)
> + goto out;
> +
> + error = -EFAULT;
> + if (copy_from_user(&new_limits, limits, sizeof(new_limits)))
> + goto out;
> +
> + error = -EINVAL;
> + if (new_limits[0] > UB_MAXVALUE || new_limits[1] > UB_MAXVALUE)
> + goto out;
> +
> + error = -ENOENT;
> + ub = beancounter_findcreate(uid, NULL, 0);
> + if (ub == NULL)
> + goto out;
> +
> + spin_lock_irqsave(&ub->ub_lock, flags);
> + ub->ub_parms[resource].barrier = new_limits[0];

> + ub->ub_parms[resource].limit = new_limits[1];
> + spin_unlock_irqrestore(&ub->ub_lock, flags);
> +
> + put_beancounter(ub);
> + error = 0;
> +out:
> + return error;
> +}
> +#endif
>
>

Subject: Re: [ckrm-tech] [RFC][PATCH 3/7] UBC: ub context and inheritance
Posted by Matt Helsley on Fri, 18 Aug 2006 02:42:51 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-16 at 19:38 +0400, Kirill Korotaev wrote:
> Contains code responsible for setting UB on task,
> it's inheriting and setting host context in interrupts.
>
> Task references three beancounters:
> 1. exec_ub current context. all resources are
> charged to this beancounter.

nit: 2-3 below seem to contradict "all". If you mean "the rest" then
perhaps you ought to reorder these:

1. task_ub ...
2. fork_sub ...
3. exec_ub Current context. Resources not charged to task_ub
or fork_sub are charged to this beancounter.

> 2. task_ub beancounter to which task_struct is
> charged itself.

Is task_ub frequently the parent beancounter of exec_ub? If it's always
the parent then perhaps the one or more of these _ub fields in the task
struct are not necessary. Also in that case keeping copies of the
"parent" user_beancounter pointers in the task_beancounters would seem
bug-prone -- if the hierarchy of beancounters changes then these would

need to be changed too.

> 3. fork_sub beancounter which is inherited by
> task's children on fork

Is this frequently the same as exec_ub?

> Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
> Signed-Off-By: Kirill Korotaev <dev@sw.ru>
>
> ---
> include/linux/sched.h | 5 +++++
> include/ub/task.h | 42 ++
> kernel/fork.c | 21 +++++++++++++++++++-----
> kernel/irq/handle.c | 9 +++++++++
> kernel/softirq.c | 8 ++++++++
> kernel/ub/Makefile | 1 +
> kernel/ub/beancounter.c | 4 ++++
> kernel/ub/misc.c | 34 ++++++++++++++++++++++++++++++++++
> 8 files changed, 119 insertions(+), 5 deletions(-)
>
> --- ./include/linux/sched.h.ubfork 2006-07-17 17:01:12.000000000 +0400
> +++ ./include/linux/sched.h 2006-07-31 16:01:54.000000000 +0400
> @@ -81,6 +81,8 @@ struct sched_param {
> #include <linux/timer.h>
> #include <linux/hrtimer.h>
>
> +#include <ub/task.h>
> +
> #include <asm/processor.h>
>
> struct exec_domain;
> @@ -997,6 +999,9 @@ struct task_struct {
> spinlock_t delays_lock;
> struct task_delay_info *delays;
> #endif
> +#ifdef CONFIG_USER_RESOURCE
> + struct task_beancounter task_bc;
> +#endif
> };
>
> static inline pid_t process_group(struct task_struct *tsk)
> --- ./include/ub/task.h.ubfork 2006-07-28 18:53:52.000000000 +0400
> +++ ./include/ub/task.h 2006-08-01 15:26:08.000000000 +0400
> @@ -0,0 +1,42 @@
> +/*
> + * include/ub/task.h
> + *

> + * Copyright (C) 2006 OpenVZ. SWsoft Inc
> + *
> + */
> +
> +#ifndef __UB_TASK_H_
> +#define __UB_TASK_H_
> +
> +#include <linux/config.h>
> +
> +struct user_beancounter;
> +
> +struct task_beancounter {
> + struct user_beancounter *exec_ub;
> + struct user_beancounter *task_ub;
> + struct user_beancounter *fork_sub;
> +};
> +
> +#ifdef CONFIG_USER_RESOURCE
> +#define get_exec_ub() (current->task_bc.exec_ub)
> +#define set_exec_ub(newub)   \
> + ({   \
> +   struct user_beancounter *old; \
> +   struct task_beancounter *tbc; \
> +   tbc = &current->task_bc; \
> +   old = tbc->exec_ub; \
> +   tbc->exec_ub = newub; \
> +   old;   \
> + })
> +

How about making these static inlines?

> +int ub_task_charge(struct task_struct *parent, struct task_struct *new);
> +void ub_task_uncharge(struct task_struct *tsk);
> +
> +#else /* CONFIG_USER_RESOURCE */
> +#define get_exec_ub() (NULL)
> +#define set_exec_ub(__ub) (NULL)
> +#define ub_task_charge(p, t) (0)
> +#define ub_task_uncharge(t) do { } while (0)
> +#endif /* CONFIG_USER_RESOURCE */
> +#endif /* __UB_TASK_H_ */
> --- ./kernel/irq/handle.c.ubirq 2006-07-10 12:39:20.000000000 +0400
> +++ ./kernel/irq/handle.c 2006-08-01 12:39:34.000000000 +0400
> @@ -16,6 +16,9 @@
> #include <linux/interrupt.h>
> #include <linux/kernel_stat.h>
>

> +#include <ub/beancounter.h>
> +#include <ub/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 user_beancounter *ub;
> +
> + ub = set_exec_ub(&ub0);

Perhaps a comment: "/* Don't charge resources gained in interrupts to current */

>   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);
> +
> + (void) set_exec_ub(ub);
>   return 1;
>   }
>
> @@ -246,6 +254,7 @@ out:
>   desc->chip->end(irq);
>   spin_unlock(&desc->lock);
>
> + (void) set_exec_ub(ub);

Seems like a WARN_ON() would be appropriate rather than ignoring the
return code.

>   return 1;
> }
>
> --- ./kernel/softirq.c.ubirq 2006-07-17 17:01:12.000000000 +0400
> +++ ./kernel/softirq.c 2006-08-01 12:40:44.000000000 +0400
> @@ -18,6 +18,9 @@
> #include <linux/rcupdate.h>
> #include <linux/smp.h>
>
> +#include <ub/beancounter.h>
> +#include <ub/task.h>

> +
> #include <asm/irq.h>
> /*
>   - No shared variables, all the data are CPU local.
> @@ -191,6 +194,9 @@ asmlinkage void __do_softirq(void)
>   __u32 pending;
>   int max_restart = MAX_SOFTIRQ_RESTART;
>   int cpu;
> + struct user_beancounter *ub;
> +
> + ub = set_exec_ub(&ub0);

Perhaps add the same comment...

>   pending = local_softirq_pending();
>   account_system_vtime(current);
> @@ -229,6 +235,8 @@ restart:
>
>   account_system_vtime(current);
>   _local_bh_enable();
> +
> + (void) set_exec_ub(ub);

.. and the same WARN_ON.

> }
>
> #ifndef __ARCH_HAS_DO_SOFTIRQ
> --- ./kernel/fork.c.ubfork 2006-07-17 17:01:12.000000000 +0400
> +++ ./kernel/fork.c 2006-08-01 12:58:36.000000000 +0400
> @@ -46,6 +46,8 @@
> #include <linux/delayacct.h>
> #include <linux/taskstats_kern.h>
>
> +#include <ub/task.h>
> +
> #include <asm/pgtable.h>
> #include <asm/pgalloc.h>
> #include <asm/uaccess.h>
> @@ -102,6 +104,7 @@ static kmem_cache_t *mm_cachep;
>
> void free_task(struct task_struct *tsk)
> {
> + ub_task_uncharge(tsk);
>   free_thread_info(tsk->thread_info);
>   rt_mutex_debug_task_free(tsk);
>   free_task_struct(tsk);
> @@ -162,18 +165,19 @@ static struct task_struct *dup_task_stru

>
>   tsk = alloc_task_struct();
>   if (!tsk)
> - return NULL;
> + goto out;
>
>   ti = alloc_thread_info(tsk);
> - if (!ti) {
> - free_task_struct(tsk);
> - return NULL;
> - }
> + if (!ti)
> + goto out_tsk;
>
>   *tsk = *orig;
>   tsk->thread_info = ti;
>   setup_thread_stack(tsk, orig);
>
> + if (ub_task_charge(orig, tsk))
> + goto out_ti;
> +
>   /* One for us, one for whoever does the "release_task()" (usually parent) */
>   atomic_set(&tsk->usage,2);
>   atomic_set(&tsk->fs_excl, 0);
> @@ -180,6 +184,13 @@ static struct task_struct *dup_task_stru
> #endif
>   tsk->splice_pipe = NULL;
>   return tsk;
> +
> +out_ti:
> + free_thread_info(ti);
> +out_tsk:
> + free_task_struct(tsk);
> +out:
> + return NULL;

Ugh. This is starting to look like copy_process(). Any reason you
couldn't move the bean counter bits to copy_process() instead?

> }
>
> #ifdef CONFIG_MMU
> --- ./kernel/ub/Makefile.ubcore 2006-08-03 16:24:56.000000000 +0400
> +++ ./kernel/ub/Makefile 2006-08-01 11:08:39.000000000 +0400
> @@ -5,3 +5,4 @@
> #
>
> obj-$(CONFIG_USER_RESOURCE) += beancounter.o

> +obj-$(CONFIG_USER_RESOURCE) += misc.o
> --- ./kernel/ub/beancounter.c.ubcore 2006-07-28 13:07:44.000000000 +0400
> +++ ./kernel/ub/beancounter.c 2006-08-03 16:14:17.000000000 +0400
> @@ -395,6 +395,10 @@
> spin_lock_init(&ub_hash_lock);
> slot = &ub_hash[ub_hash_fun(ub->ub_uid)];
> hlist_add_head(&ub->hash, slot);
> +
> + current->task_bc.exec_ub = ub;
> + current->task_bc.task_ub = get_beancounter(ub);
> + current->task_bc.fork_sub = get_beancounter(ub);
> }
>
> void __init ub_init_late(void)
> --- ./kernel/ub/misc.c.ubfork 2006-07-31 16:23:44.000000000 +0400
> +++ ./kernel/ub/misc.c 2006-07-31 16:28:47.000000000 +0400
> @@ -0,0 +1,34 @@
> +/*
> + * kernel/ub/misc.c
> + *
> + * Copyright (C) 2006 OpenVZ. SWsoft Inc.
> + *
> + */
> +
> +#include <linux/sched.h>
> +
> +#include <ub/beancounter.h>
> +#include <ub/task.h>
> +
> +int ub_task_charge(struct task_struct *parent, struct task_struct *new)
> +{

parent could be derived from new if you move the charge to copy_process
instead of dup_task_struct.

> + struct task_beancounter *old_bc;
> + struct task_beancounter *new_bc;
> + struct user_beancounter *ub;
> +
> + old_bc = &parent->task_bc;
> + new_bc = &new->task_bc;
> +
> + ub = old_bc->fork_sub;
> + new_bc->exec_ub = get_beancounter(ub);
> + new_bc->task_ub = get_beancounter(ub);
> + new_bc->fork_sub = get_beancounter(ub);
> + return 0;
> +}

> +
> +void ub_task_uncharge(struct task_struct *tsk)
> +{
> + put_beancounter(tsk->task_bc.exec_ub);
> + put_beancounter(tsk->task_bc.task_ub);
> + put_beancounter(tsk->task_bc.fork_sub);
> +}
>

Subject: Re: [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Andrew Morton on Fri, 18 Aug 2006 05:31:37 GMT
View Forum Message <> Reply to Message

On Thu, 17 Aug 2006 15:53:40 +0400
Kirill Korotaev <dev@sw.ru> wrote:

> >>+struct user_beancounter
> >>+{
> >>+ atomic_t ub_refcount;
> >>+ spinlock_t ub_lock;
> >>+ uid_t ub_uid;
> >>
> >
> > Why uid? Will it be possible to club processes belonging to different
> > users to same bean counter.
> oh, its a misname. Should be ub_id. it is ID of user_beancounter
> and has nothing to do with user id.

But it uses a uid_t. That's more than a misnaming?

Subject: Re: [PATCH 2/7] UBC: core (structures, API)
Posted by Andrey Savochkin on Fri, 18 Aug 2006 07:35:25 GMT
View Forum Message <> Reply to Message

On Thu, Aug 17, 2006 at 10:31:37PM -0700, Andrew Morton wrote:
> On Thu, 17 Aug 2006 15:53:40 +0400
> Kirill Korotaev <dev@sw.ru> wrote:
>

> > >>+struct user_beancounter
> > >>+{
> > >>+ atomic_t ub_refcount;
> > >>+ spinlock_t ub_lock;
> > >>+ uid_t   ub_uid;
> > >
> > >
> > > Why uid? Will it be possible to club processes belonging to different
> > > users to same bean counter.
> > oh, its a misname. Should be ub_id. it is ID of user_beancounter
> > and has nothing to do with user id.
>
> But it uses a uid_t. That's more than a misnaming?

It used to be uid-related in ancient times when the notion of container
hadn't formed up.
"user" part of user_beancounter name has the same origin :)

Now ub_id_t or something like that would be the most logical type.

 Andrey

Subject: Re: [PATCH 4/7] UBC: syscalls (user interface)
Posted by Andrey Savochkin on Fri, 18 Aug 2006 08:08:09 GMT
View Forum Message <> Reply to Message

Hi,

On Thu, Aug 17, 2006 at 08:40:33AM -0700, Andrew Morton wrote:
> On Thu, 17 Aug 2006 16:13:30 +0400
> Kirill Korotaev <dev@sw.ru> wrote:
>
> > > I was more thinking about (for example) user land physical memory limit
> > > for that bean counter. If the limits are going down, then the system
> > > call should try to flush out page cache pages or swap out anonymous
> > > memory. But you are right that it won't be possible in all cases, like
> > > for in kernel memory limits.
> > Such kind of memory management is less efficient than the one
> > making decisions based on global shortages and global LRU alogrithm.
>
> I also was quite surprised that openvz appears to have no way of
> constraining a container's memory usage. "I want to run this bunch of
> processes in a 4.5GB container".

I'd like to share my view on the subject of memory usage limiting.

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.

Best regards

Andrey

Subject: Re: [ckrm-tech] [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by dev on Fri, 18 Aug 2006 08:12:57 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Thu, 2006-08-17 at 17:31 +0400, Kirill Korotaev wrote:
>
>>>How many things actually use this? Can we have the slab ubcs
>>
>>without
>>
>>>the struct page pointer?
>>
>>slab doesn't use this pointer on the page.
>>It is used for pages allocated by buddy
>>alocator implicitly (e.g. LDT pages, page tables, ...).
>
>
> Hmmm. There aren't _that_ many of those cases, right? Are there any
> that absolutely need raw access to the buddy allocator? I'm pretty sure
> that pagetables can be moved over to a slab, as long as we bump up the
> alignment.
LDT takes from 1 to 16 pages. and is allocated by vmalloc.
do you propose to replace it with slab which can fail due to memory
fragmentation?

the same applies to fdset, fdarray, ipc ids and iptables entries.

> It does seem a wee bit silly to have the pointer in _all_ of the struct
> pages, even the ones for which we will never do any accounting (and even
> on kernels that never need it). But, a hashing scheme sounds like a
> fine idea.
It seems a silly for you since 2nd patchset accounting user pages
is not here yet. As you can see we added a union into page,
which is shared between kernel memory and user memory accounting.

THERE IS NOT USER ACCOUNTING HERE YET GUYS! :) THIS FIELD WILL BE USED!!!

Thanks,
Kirill

Subject: Re: [ckrm-tech] [PATCH 2/7] UBC: core (structures, API)
Posted by Matt Helsley on Fri, 18 Aug 2006 08:26:03 GMT
View Forum Message <> Reply to Message

On Fri, 2006-08-18 at 11:35 +0400, Andrey Savochkin wrote:
> On Thu, Aug 17, 2006 at 10:31:37PM -0700, Andrew Morton wrote:
> > On Thu, 17 Aug 2006 15:53:40 +0400
> > Kirill Korotaev <dev@sw.ru> wrote:
> >
> > > > >+struct user_beancounter

> > > >>+{
> > > >>+ atomic_t ub_refcount;
> > > >>+ spinlock_t ub_lock;
> > > >>+ uid_t   ub_uid;
> > > >
> > > >
> > > > Why uid? Will it be possible to club processes belonging to different
> > > > users to same bean counter.
> > > oh, its a misname. Should be ub_id. it is ID of user_beancounter
> > > and has nothing to do with user id.
> >
> > But it uses a uid_t. That's more than a misnaming?
>
> It used to be uid-related in ancient times when the notion of container
> hadn't formed up.
> "user" part of user_beancounter name has the same origin :)

Is it similarly irrelevant now? If so perhaps a big rename could be used
to make the names clearer (s/user_//, s/ub_/bc_/, ...).

<snip>

Cheers,
 -Matt Helsley

Subject: Re: [ckrm-tech] [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by dev on Fri, 18 Aug 2006 08:43:45 GMT
View Forum Message <> Reply to Message

Rohit Seth wrote:
> On Thu, 2006-08-17 at 01:22 +0100, Alan Cox wrote:
>
>>Ar Mer, 2006-08-16 am 12:15 -0700, ysgrifennodd Rohit Seth:
>>
>>>resources will be allocated/freed in context of a user process. And at
>>>that time we know if a allocation should succeed or not. So we may
>>>actually not need to track kernel pages that closely.
>>
>>Quite the reverse, tracking kernel pages is critical,
>
>
> Having the knowledge of how many kernel pages are getting used by each
> container is indeed very useful. But as long as the context in which
> they are created and destroyed is identifiable, there is no need to
> really physically tag each page with container id. And for the cases
> where we have no context, it will be worth while to see if mapping field
> could be used.

as I described in another email this field is also reused for tracking
user pages.

Kirill

Subject: Re: [ckrm-tech] [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by dev on Fri, 18 Aug 2006 08:47:57 GMT
View Forum Message <> Reply to Message

Rohit Seth wrote:
> On Thu, 2006-08-17 at 17:35 +0400, Kirill Korotaev wrote:
>
>
>>>My preference would be to have container (I keep on saying container,
>>>but resource beancounter) pointer embeded in task, mm(not sure),
>>>address_space and anon_vma structures. This should allow us to track
>>>user land pages optimally. But for tracking kernel usage on behalf of
>>>user, we will have to use an additional field (unless we can re-use
>>>mapping). Please correct me if I'm wrong, though all the kernel
>>>resources will be allocated/freed in context of a user process. And at
>>>that time we know if a allocation should succeed or not. So we may
>>>actually not need to track kernel pages that closely. We are not going
>>>to run reclaim on any of them anyways.
>>
>>objects are really allocated in process context
>>(except for TCP/IP and other softirqs which are done in arbitrary
>>process context!)
>
>
> Can these pages be tagged using mapping field of the page struct.
kernel pages can be taged with mapping field.
User pages - not. So we introduce 2 pointers in the unoin:
union {
  page_ub // for kernel pages
  page_pb // for user pages
}

>
>
>>And objects are not always freed in correct context (!).
>>
>
> You mean beyond Networking and softirq.
>
>
>>Note, page_ub is not for _user_ pages. user pages accounting will be added
>>in next patch set. page_ub is added to track kernel allocations.

>>
>
>
> But will the page_ub be used for some purpose for user land pages?
yes. see above.

Kirill

Subject: Re: [ckrm-tech] [RFC][PATCH 3/7] UBC: ub context and inheritance
Posted by dev on Fri, 18 Aug 2006 09:21:22 GMT
View Forum Message <> Reply to Message

Matt Helsley wrote:
> On Wed, 2006-08-16 at 19:38 +0400, Kirill Korotaev wrote:
>
>>Contains code responsible for setting UB on task,
>>it's inheriting and setting host context in interrupts.
>>
>>Task references three beancounters:
>> 1. exec_ub current context. all resources are
>>   charged to this beancounter.
>
>
> nit: 2-3 below seem to contradict "all". If you mean "the rest" then
> perhaps you ought to reorder these:
>
> 1. task_ub ...
> 2. fork_sub ...
> 3. exec_ub Current context. Resources not charged to task_ub
>   or fork_sub are charged to this beancounter.
not sure what you mean.
task_ub - where _task_ _itself_ is charged as an object.
        following patches will add charging of "number of tasks" using it.
fork_sub - beancounter which is inherited on fork() (chaning task beancounter).
exec_ub - is current context.

>> 2. task_ub beancounter to which task_struct is
>>   charged itself.
>
>
> Is task_ub frequently the parent beancounter of exec_ub? If it's always
> the parent then perhaps the one or more of these _ub fields in the task
> struct are not necessary.
no, task_ub != exec_ub of parent task
when task is created anything can happen: task can change ub, parent can change ub,
task can be reparented. But the UB we charged task to should be known.

> Also in that case keeping copies of the
> "parent" user_beancounter pointers in the task_beancounters would seem
> bug-prone -- if the hierarchy of beancounters changes then these would
> need to be changed too.
>
>
>> 3. fork_sub beancounter which is inherited by
>>   task's children on fork
>
>
> Is this frequently the same as exec_ub?
frequently, but not always. exec_ub is changed in softirq for example.
consider exec_ub as 'current' pointer in kernel.

see other comments below

>>Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
>>Signed-Off-By: Kirill Korotaev <dev@sw.ru>
>>
>>---
>> include/linux/sched.h | 5 +++++
>> include/ub/task.h | 42 ++
>> kernel/fork.c | 21 +++++++++++++++++-----
>> kernel/irq/handle.c | 9 +++++++++
>> kernel/softirq.c | 8 ++++++++
>> kernel/ub/Makefile | 1 +
>> kernel/ub/beancounter.c | 4 ++++
>> kernel/ub/misc.c | 34 ++++++++++++++++++++++++++++++++++
>> 8 files changed, 119 insertions(+), 5 deletions(-)
>>
>>--- ./include/linux/sched.h.ubfork 2006-07-17 17:01:12.000000000 +0400
>>+++ ./include/linux/sched.h 2006-07-31 16:01:54.000000000 +0400
>>@@ -81,6 +81,8 @@ struct sched_param {
>> #include <linux/timer.h>
>> #include <linux/hrtimer.h>
>>
>>+#include <ub/task.h>
>>+
>> #include <asm/processor.h>
>>
>> struct exec_domain;
>>@@ -997,6 +999,9 @@ struct task_struct {
>> spinlock_t delays_lock;
>> struct task_delay_info *delays;
>> #endif
>>+#ifdef CONFIG_USER_RESOURCE
>>+ struct task_beancounter task_bc;

>>+#endif
>> };
>>
>> static inline pid_t process_group(struct task_struct *tsk)
>>--- ./include/ub/task.h.ubfork 2006-07-28 18:53:52.000000000 +0400
>>+++ ./include/ub/task.h 2006-08-01 15:26:08.000000000 +0400
>>@@ -0,0 +1,42 @@
>>+/*
>>+ * include/ub/task.h
>>+ *
>>+ * Copyright (C) 2006 OpenVZ. SWsoft Inc
>>+ *
>>+ */
>>+
>>+#ifndef __UB_TASK_H_
>>+#define __UB_TASK_H_
>>+
>>+#include <linux/config.h>
>>+
>>+struct user_beancounter;
>>+
>>+struct task_beancounter {
>>+ struct user_beancounter *exec_ub;
>>+ struct user_beancounter *task_ub;
>>+ struct user_beancounter *fork_sub;
>>+};
>>+
>>+#ifdef CONFIG_USER_RESOURCE
>>+#define get_exec_ub() (current->task_bc.exec_ub)
>>+#define set_exec_ub(newub)   \
>>+ ({   \
>>+   struct user_beancounter *old; \
>>+   struct task_beancounter *tbc; \
>>+   tbc = &current->task_bc; \
>>+   old = tbc->exec_ub; \
>>+   tbc->exec_ub = newub; \
>>+   old;   \
>>+ })
>>+
>
>
> How about making these static inlines?
possible, but this requires including sched.h, which includes this file...
so this one is easier and more separated.

>>+int ub_task_charge(struct task_struct *parent, struct task_struct *new);
>>+void ub_task_uncharge(struct task_struct *tsk);
>>+

>>+#else /* CONFIG_USER_RESOURCE */
>>+#define get_exec_ub() (NULL)
>>+#define set_exec_ub(__ub) (NULL)
>>+#define ub_task_charge(p, t) (0)
>>+#define ub_task_uncharge(t) do { } while (0)
>>+#endif /* CONFIG_USER_RESOURCE */
>>+#endif /* __UB_TASK_H_ */
>>--- ./kernel/irq/handle.c.ubirq 2006-07-10 12:39:20.000000000 +0400
>>+++ ./kernel/irq/handle.c 2006-08-01 12:39:34.000000000 +0400
>>@@ -16,6 +16,9 @@
>> #include <linux/interrupt.h>
>> #include <linux/kernel_stat.h>
>>
>>+#include <ub/beancounter.h>
>>+#include <ub/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 user_beancounter *ub;
>>+
>>+ ub = set_exec_ub(&ub0);
>
>
> Perhaps a comment: "/* Don't charge resources gained in interrupts to current */
ok, will add comment:
/* UBC charges should be done to host system */
>
>
>> 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);
>>+
>>+ (void) set_exec_ub(ub);
>>   return 1;
>> }
>>
>>@@ -246,6 +254,7 @@ out:
>> desc->chip->end(irq);
>> spin_unlock(&desc->lock);
>>

>>+ (void) set_exec_ub(ub);
>
>
>
> Seems like a WARN_ON() would be appropriate rather than ignoring the
> return code.
BUG_ON(ret != &ub0) ?

maybe introduce a kind of
reset_exec_ub(old_ub, expected_current_ub)
{
  ret = set_exec_ub(old_ub);
  BUG_ON(ret != expected_current_ub);
}
?

>> return 1;
>> }
>>
>>--- ./kernel/softirq.c.ubirq 2006-07-17 17:01:12.000000000 +0400
>>+++ ./kernel/softirq.c 2006-08-01 12:40:44.000000000 +0400
>>@@ -18,6 +18,9 @@
>> #include <linux/rcupdate.h>
>> #include <linux/smp.h>
>>
>>+#include <ub/beancounter.h>
>>+#include <ub/task.h>
>>+
>> #include <asm/irq.h>
>> /*
>>   - No shared variables, all the data are CPU local.
>>@@ -191,6 +194,9 @@ asmlinkage void __do_softirq(void)
>> __u32 pending;
>> int max_restart = MAX_SOFTIRQ_RESTART;
>> int cpu;
>>+ struct user_beancounter *ub;
>>+
>>+ ub = set_exec_ub(&ub0);
>
>
> Perhaps add the same comment...
ok

>
>
>> pending = local_softirq_pending();
>> account_system_vtime(current);

>>@@ -229,6 +235,8 @@ restart:
>>
>> account_system_vtime(current);
>> _local_bh_enable();
>>+
>>+ (void) set_exec_ub(ub);
>
>
> .. and the same WARN_ON.
>
>
>> }
>>
>> #ifndef __ARCH_HAS_DO_SOFTIRQ
>>--- ./kernel/fork.c.ubfork 2006-07-17 17:01:12.000000000 +0400
>>+++ ./kernel/fork.c 2006-08-01 12:58:36.000000000 +0400
>>@@ -46,6 +46,8 @@
>> #include <linux/delayacct.h>
>> #include <linux/taskstats_kern.h>
>>
>>+#include <ub/task.h>
>>+
>> #include <asm/pgtable.h>
>> #include <asm/pgalloc.h>
>> #include <asm/uaccess.h>
>>@@ -102,6 +104,7 @@ static kmem_cache_t *mm_cachep;
>>
>> void free_task(struct task_struct *tsk)
>> {
>>+ ub_task_uncharge(tsk);
>> free_thread_info(tsk->thread_info);
>> rt_mutex_debug_task_free(tsk);
>> free_task_struct(tsk);
>>@@ -162,18 +165,19 @@ static struct task_struct *dup_task_stru
>>
>> tsk = alloc_task_struct();
>> if (!tsk)
>>- return NULL;
>>+ goto out;
>>
>> ti = alloc_thread_info(tsk);
>>- if (!ti) {
>>- free_task_struct(tsk);
>>- return NULL;
>>- }
>>+ if (!ti)
>>+ goto out_tsk;
>>

>> *tsk = *orig;
>> tsk->thread_info = ti;
>> setup_thread_stack(tsk, orig);
>>
>>+ if (ub_task_charge(orig, tsk))
>>+ goto out_ti;
>>+
>> /* One for us, one for whoever does the "release_task()" (usually parent) */
>> atomic_set(&tsk->usage,2);
>> atomic_set(&tsk->fs_excl, 0);
>>@@ -180,6 +184,13 @@ static struct task_struct *dup_task_stru
>> #endif
>> tsk->splice_pipe = NULL;
>> return tsk;
>>+
>>+out_ti:
>>+ free_thread_info(ti);
>>+out_tsk:
>>+ free_task_struct(tsk);
>>+out:
>>+ return NULL;
>
>
> Ugh. This is starting to look like copy_process(). Any reason you
> couldn't move the bean counter bits to copy_process() instead?
This is more logical place since we _will_ charge task here
(next patchset for numproc).
It is logically better to charge objects in places where
they are allocated. At the same time we inherit tasks ubs here.

>> }
>>
>> #ifdef CONFIG_MMU
>>--- ./kernel/ub/Makefile.ubcore 2006-08-03 16:24:56.000000000 +0400
>>+++ ./kernel/ub/Makefile 2006-08-01 11:08:39.000000000 +0400
>>@@ -5,3 +5,4 @@
>> #
>>
>> obj-$(CONFIG_USER_RESOURCE) += beancounter.o
>>+obj-$(CONFIG_USER_RESOURCE) += misc.o
>>--- ./kernel/ub/beancounter.c.ubcore 2006-07-28 13:07:44.000000000 +0400
>>+++ ./kernel/ub/beancounter.c 2006-08-03 16:14:17.000000000 +0400
>>@@ -395,6 +395,10 @@
>> spin_lock_init(&ub_hash_lock);
>> slot = &ub_hash[ub_hash_fun(ub->ub_uid)];
>> hlist_add_head(&ub->hash, slot);
>>+
>>+ current->task_bc.exec_ub = ub;

>>+ current->task_bc.task_ub = get_beancounter(ub);
>>+ current->task_bc.fork_sub = get_beancounter(ub);
>> }
>>
>> void __init ub_init_late(void)
>>--- ./kernel/ub/misc.c.ubfork 2006-07-31 16:23:44.000000000 +0400
>>+++ ./kernel/ub/misc.c 2006-07-31 16:28:47.000000000 +0400
>>@@ -0,0 +1,34 @@
>>+/*
>>+ * kernel/ub/misc.c
>>+ *
>>+ * Copyright (C) 2006 OpenVZ. SWsoft Inc.
>>+ *
>>+ */
>>+
>>+#include <linux/sched.h>
>>+
>>+#include <ub/beancounter.h>
>>+#include <ub/task.h>
>>+
>>+int ub_task_charge(struct task_struct *parent, struct task_struct *new)
>>+{
>
>
> parent could be derived from new if you move the charge to copy_process
> instead of dup_task_struct.
we can split it into:
ub_charge_task() in dup_task_struct to account _task_ itself.
ub_copy_process() in copy_process() to inherit and initialize
  exec_ub and fork_sub

what do you think?

>>+ struct task_beancounter *old_bc;
>>+ struct task_beancounter *new_bc;
>>+ struct user_beancounter *ub;
>>+
>>+ old_bc = &parent->task_bc;
>>+ new_bc = &new->task_bc;
>>+
>>+ ub = old_bc->fork_sub;
>>+ new_bc->exec_ub = get_beancounter(ub);
>>+ new_bc->task_ub = get_beancounter(ub);
>>+ new_bc->fork_sub = get_beancounter(ub);
>>+ return 0;
>>+}
>>+
>>+void ub_task_uncharge(struct task_struct *tsk)

>>+{
>>+ put_beancounter(tsk->task_bc.exec_ub);
>>+ put_beancounter(tsk->task_bc.task_ub);
>>+ put_beancounter(tsk->task_bc.fork_sub);
>>+}
>> --- ----------
>>Using Tomcat but need to do more? Need to support web services, security?
>>Get stuff done quickly with pre-integrated technology to make your job easier
>>Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
>> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
>>_____
>>ckrm-tech mailing list
>>https://lists.sourceforge.net/lists/listinfo/ckrm-tech
>
>
>

Subject: Re: [ckrm-tech] [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by dev on Fri, 18 Aug 2006 09:29:26 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Thu, 2006-08-17 at 17:27 +0400, Kirill Korotaev wrote:
>
>>charged kernel objects can't be _reclaimed_. how do you propose
>>to reclaim tasks page tables or files or task struct or vma or etc.?
>
>
> Do you have any statistics on which of these objects are the most
> troublesome? If it _is_ pagetables, for instance, it is quite
> conceivable that we could reclaim them.
they all are troublesome :/
user can create lots of vmas, w/o page tables.
lots of fdsets, ipcids.
These are not reclaimable.

Also consider the following scenario with reclaimable page tables.
e.g. user hit kmemsize limit due to fat page tables.
kernel reclaims some of the page tables and frees user kenerl memory.
after that user creates some uncreclaimable objects like fdsets or ipcs
and then accesses memory with reclaimed page tables.
Sooner or later we kill user with SIGSEGV from page fault due to
no memory. This is worse then returning ENOMEM from poll() or
mmap() where user allocates kernel objects.

> This one probably deserves a big, fat comment, though. ;)
tell me where to write it and what? :)

Thanks,
Kirill

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by dev on Fri, 18 Aug 2006 09:36:19 GMT
View Forum Message <> Reply to Message

Rohit Seth wrote:
> On Thu, 2006-08-17 at 17:27 +0400, Kirill Korotaev wrote:
>
>>>If I'm reading this patch right then seems like you are making page
>>>allocations to fail w/o (for example) trying to purge some pages from
>>>the page cache belonging to this container. Or is that reclaim going to
>>>come later?
>>
>>charged kernel objects can't be _reclaimed_. how do you propose
>>to reclaim tasks page tables or files or task struct or vma or etc.?
>
>
>
> I agree that kernel objects cann't be reclaimed easily. But what you
> are proposing is also not right. Returning failure w/o doing any
> reclaim on pages (that are reclaimable) is not useful. And this is why
> I asked, is this change going to be part of next set of patches (as
> current set of patches are only tracking kernel usage).
1. reclaiming user resources is not that good idea as it looks to you.
such solutions end up with lots of resources spent on reclaim.
for user memory reclaims mean consumption of expensive disk I/O bandwidth
which reduces overall system throughput and influences other users.

2. kernel memory is mostly not reclaimable. can you reclaim vma structs or ipc ids?
even with page tables it is not that easy.
And the fact is:
 - kernel memory consumtion is usually less then user memory,
   so it's not worth reclaiming it.
 - reclaiming can result in kind of user service deadlocks when you are
   unable to handle user requests gracefully anymore.
   See my email to Dave with an example.
 - our solution _is_ right and works (for >3 years in production already).
   If desired it _can_ be extended with reclamation.

Kirill

Subject: Re: [RFC][PATCH 4/7] UBC: syscalls (user interface)

Andrew Morton wrote:
> On Thu, 17 Aug 2006 16:13:30 +0400
> Kirill Korotaev <dev@sw.ru> wrote:
>
>
>>>I was more thinking about (for example) user land physical memory limit
>>>for that bean counter. If the limits are going down, then the system
>>>call should try to flush out page cache pages or swap out anonymous
>>>memory. But you are right that it won't be possible in all cases, like
>>>for in kernel memory limits.
>>
>>Such kind of memory management is less efficient than the one
>>making decisions based on global shortages and global LRU alogrithm.
>
>
> I also was quite surprised that openvz appears to have no way of
> constraining a container's memory usage. "I want to run this bunch of
> processes in a 4.5GB container".
If you mean user memory, then it is possible to set
container limits to 4,5GB. This is what most people care about
and it is not a problem.

Or you mean that you are suprised there are lots of parameters
and there is no a single one limiting the _whole_ memory set of container
memory (sum of kernel memory, user space memory and other resources memory)?

>>The problem here is that doing swap out takes more expensive disk I/O
>>influencing other users.
>
>
> A well-set-up container would presumably be working against its own
> spindle(s). If the operator has gone to all the trouble of isolating a job
> from the system's other jobs, he'd be pretty dumb to go and let all the
> "isolated" jobs share a stinky-slow resource like a disk.
why do you assume that it is always an operator who controls the applications
inside the container?
users can run any application inside and it is systems job to
introduce resource isolation between users, not the operators full-time
job doing monitoring of users.

> But yes, swap is a problem. To do this properly we'd need a way of saying
> "this container here uses that swap device over there".
yep, this is possible with page beancounters as it tracks user pages.
more over, we have an intention of building a system with a single container
memory parameter, but we think this is more user interface question and
still requires all the UBC resources accounting.

Thanks,
Kirill

Subject: Re: [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by dev on Fri, 18 Aug 2006 11:13:04 GMT
View Forum Message <> Reply to Message

Rohit Seth wrote:
> On Thu, 2006-08-17 at 15:53 +0400, Kirill Korotaev wrote:
>
>>Rohit Seth wrote:
>>
>>>On Wed, 2006-08-16 at 19:37 +0400, Kirill Korotaev wrote:
>>>
>>>
>>>>Core functionality and interfaces of UBC:
>>>>find/create beancounter, initialization,
>>>>charge/uncharge of resource, core objects' declarations.
>>>>
>>>>Basic structures:
>>>> ubparm - resource description
>>>> user_beancounter - set of resources, id, lock
>>>>
>>>>Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
>>>>Signed-Off-By: Kirill Korotaev <dev@sw.ru>
>>>>
>>>>---
>>>>include/ub/beancounter.h | 157 ++++++++++++++++++
>>>>init/main.c | 4
>>>>kernel/Makefile | 1
>>>>kernel/ub/Makefile | 7
>>>>kernel/ub/beancounter.c | 398
++
>>>>5 files changed, 567 insertions(+)
>>>>
>>>>--- /dev/null 2006-07-18 14:52:43.075228448 +0400
>>>>+++ ./include/ub/beancounter.h 2006-08-10 14:58:27.000000000 +0400
>>>>@@ -0,0 +1,157 @@
>>>>+/*
>>>>+ * include/ub/beancounter.h
>>>>+ *
>>>>+ * Copyright (C) 2006 OpenVZ. SWsoft Inc
>>>>+ *
>>>>+ */
>>>>+
>>>>+#ifndef _LINUX_BEANCOUNTER_H

```
>>>>+#define _LINUX_BEANCOUNTER_H
>>>>+
>>>>+/*
>>>>+ * Resource list.
>>>>+ */
>>>>+
>>>>+#define UB_RESOURCES 0
>>>>+
>>>>+struct ubparm {
>>>>+ /*
>>>>+  * 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 barrier;
>>>>+ /* hard resource limit */
>>>>+ unsigned long limit;
>>>>+ /* consumed resources */
>>>>+ unsigned long held;
>>>>+ /* maximum amount of consumed resources through the last period */
>>>>+ unsigned long maxheld;
>>>>+ /* minimum amount of consumed resources through the last period */
>>>>+ unsigned long minheld;
>>>>+ /* count of failed charges */
>>>>+ unsigned long failcnt;
>>>>+};
>>>
>>>
>>>What is the difference between barrier and limit. They both sound like
>>>hard limits.  No?
>>
>>check __charge_beancounter_locked and severity.
>>It provides some kind of soft and hard limits.
>>
>
>
> Would be easier to just rename them as soft and hard limits...
>
>
>>>>+
>>>>+/*
>>>>+ * Kernel internal part.
>>>>+ */
>>>>+
>>>>+#ifdef __KERNEL__
>>>>+
>>>>+#include <linux/config.h>
>>>>+#include <linux/spinlock.h>
```

>>>>+#include <linux/list.h>
>>>>+#include <asm/atomic.h>
>>>>+
>>>>+/*
>>>>+ * UB_MAXVALUE is essentially LONG_MAX declared in a cross-compiling safe form.
>>>>+ */
>>>>+ /* resources statistics and settings */
>>>>+ struct ubparm ub_parms[UB_RESOURCES];
>>>>+};
>>>>+
>>>
>>>
>>>I presume UB_RESOURCES value is going to change as different resources
>>>start getting tracked.
>>
>>what's wrong with it?
>>
>
>
> ...just that user land will need to be some how informed about that.
the same way user space knows that system call is (not) implemented.
(include unistd.h :))))

>>>I think something like configfs should be used for user interface. It
>>>automatically presents the right interfaces to user land (based on
>>>kernel implementation). And you wouldn't need any changes in glibc etc.
>>
>>1. UBC doesn't require glibc modificatins.
>
>
> You are right not for setting the limits. But for adding any new
> functionality related to containers....as in you just added a new system
> call to get the limits.
Do you state that glibc describes _all_ the existing system calls with some wrappers?

>>2. if you think a bit more about it, adding UB parameters doesn't
>>   require user space changes as well.
>>3. it is possible to add any kind of interface for UBC. but do you like the idea
>>   to grep 200(containers)x20(parameters) files for getting current usages?
>
>
> How are you doing it currently and how much more efficient it is in
> comparison to configfs?
currently it is done with a single file read.
you can grep it, sum up resources or do what ever you want from bash.
what is important! you can check whether container hits its limits
with a single command, while with configs you would have to look through
20 files...

IMHO it is convinient to have a text file representing the whole information state
and system call for applications.

>> Do you like the idea to convert numbers to strings and back w/o
>> thinking of data types?
>
>
> IMO, setting up limits and containers (themselves) is not a common
> operation. I wouldn't be too worried about loosing those few extra
> cycles in setting them up.
it is not the question of performance...

Kirill

Subject: Re: [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by Arnd Bergmann on Fri, 18 Aug 2006 11:40:30 GMT
View Forum Message <> Reply to Message

On Wednesday 16 August 2006 17:39, Kirill Korotaev wrote:

> @@ -304,3 +304,6 @@ SYSCALL_SPU(fchmodat)

> +SYSCALL(sys_getluid)
> +SYSCALL(sys_setluid)
> +SYSCALL(sys_setublimit)
...

> @@ -619,10 +619,16 @@ __SYSCALL(__NR_sync_file_range, sys_sync

> +__SYSCALL(__NR_getluid, sys_getluid)

> +__SYSCALL(__NR_setluid, sys_setluid)

> +__SYSCALL(__NR_setublimit, sys_setublimit)

...
> +/*

> + */

> +asmlinkage long sys_setublimit(uid_t uid, unsigned long resource,

While I don't yet understand what this call does, it looks to me that
the way it's implemented breaks in 32 bit emulation mode on x86_64 and
powerpc.

You either need to pass a pointer a something that is the same on 32 and
64 bit (e.g. __u64 __user *limits), or need to provide a different
entry point for 32 bit applications:

long compat_sys_setublimit(compat_uid_t uid, compat_ulong_t resource,
    compat_ulong_t __user *limits);

You should also add the prototypes to include/linux/syscalls.h.

Arnd <><

Subject: Re: [ckrm-tech] [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by dev on Fri, 18 Aug 2006 11:43:38 GMT
View Forum Message <> Reply to Message

Matt Helsley wrote:

[... snip ...]
>>--- ./kernel/ub/sys.c.ubsys 2006-07-28 18:52:18.000000000 +0400
>>+++ ./kernel/ub/sys.c 2006-08-03 16:14:23.000000000 +0400
>>@@ -0,0 +1,126 @@
>>+/*
>>+ * kernel/ub/sys.c
>>+ *
>>+ * Copyright (C) 2006 OpenVZ. SWsoft Inc
>>+ *
>>+ */
>>+
>>+#include <linux/config.h>
>>+#include <linux/sched.h>
>>+#include <asm/uaccess.h>
>>+
>>+#include <ub/beancounter.h>
>>+#include <ub/task.h>
>>+
>>+#ifndef CONFIG_USER_RESOURCE
>
>
> Get rid of the #ifdef since this file should only be compiled if
> CONFIG_USER_RESOURCE=y anyway.

>
>
>>+asmlinkage long sys_getluid(void)
>>+{
>>+ return -ENOSYS;
>>+}
>>+
>>+asmlinkage long sys_setluid(uid_t uid)
>>+{
>>+ return -ENOSYS;
>>+}
>>+
>>+asmlinkage long sys_setublimit(uid_t uid, unsigned long resource,
>>+ unsigned long *limits)
>>+{
>>+ return -ENOSYS;
>>+}
>
>
> Looks to me like you want to add:
>
> cond_syscall(sys_getluid);
> ...
>
> in kernel/sys_ni.c and then you won't have to worry about making these
> empty functions.
Good note. Thanks, will do it!

>>+#else /* CONFIG_USER_RESOURCE */
>>+
>>+/*
>>+ * The (rather boring) getluid syscall
>>+ */
>>+asmlinkage long sys_getluid(void)
>>+{
>>+ struct user_beancounter *ub;
>>+
>>+ ub = get_exec_ub();
>>+ if (ub == NULL)
>>+ return -EINVAL;
>>+
>>+ return ub->ub_uid;
>>+}
>>+
>>+/*
>>+ * The setluid syscall
>>+ */
>>+asmlinkage long sys_setluid(uid_t uid)

>>+{
>>+ int error;
>>+ struct user_beancounter *ub;
>>+ struct task_beancounter *task_bc;
>>+
>>+ task_bc = &current->task_bc;
>>+
>>+ /* You may not disown a setluid */
>>+ error = -EINVAL;
>>+ if (uid == (uid_t)-1)
>>+ goto out;
>>+
>>+ /* You may only set an ub as root */
>>+ error = -EPERM;
>>+ if (!capable(CAP_SETUID))
>>+ goto out;
>
>
> With resource groups you don't necessarily have to be root -- just the
> owner of the group and task.
the question is - who is the owner of group?
user, user group or who?
Both are bad, since the same user can run inside the container and thus
container will be potentially controllable/breakable from inside.

> Filesystems and appropriate share representations offer a way to give
> regular users the ability to manage their resources without requiring
> CAP_FOO.
not sure what you propose...

we can introduce the following rules:

containers (UB) can be created by process with SETUID cap only.
subcontainers (SUB) can be created by any process.

what do you think?

>>+ /* Ok - set up a beancounter entry for this user */
>>+ error = -ENOBUFS;
>>+ ub = beancounter_findcreate(uid, NULL, UB_ALLOC);
>>+ if (ub == NULL)
>>+ goto out;
>>+
>>+ /* install bc */
>>+ put_beancounter(task_bc->exec_ub);
>>+ task_bc->exec_ub = ub;

>>+ put_beancounter(task_bc->fork_sub);
>>+ task_bc->fork_sub = get_beancounter(ub);
>>+ error = 0;
>>+out:
>>+ return error;
>>+}
>>+
>>+/*
>>+ * The setbeanlimit syscall
>>+ */
>>+asmlinkage long sys_setublimit(uid_t uid, unsigned long resource,
>>+ unsigned long *limits)
>>+{
>>+ int error;
>>+ unsigned long flags;
>>+ struct user_beancounter *ub;
>>+ unsigned long new_limits[2];
>>+
>>+ error = -EPERM;
>>+ if(!capable(CAP_SYS_RESOURCE))
>>+ goto out;
>
>
> Again, a filesystem interface would give us more flexibility when it
> comes to allowing users to manage their resources while still preventing
> them from exceeding limits.
we can have 2 different root users with uid = 0 in 2 different containers.

> I doubt you really want to give owners of a container CAP_SYS_RESOURCE
> and CAP_USER (i.e. total control over resource management) just to allow
> them to manage their subset of the resources.
The origin idea is that administator of the node can manage user
resources only. Users can't, since otherwise they can increase the limits.
But we can allow them to manage sub beancoutners imho...

>>+ error = -EINVAL;
>>+ if (resource >= UB_RESOURCES)
>>+ goto out;
>>+
>>+ error = -EFAULT;
>>+ if (copy_from_user(&new_limits, limits, sizeof(new_limits)))
>>+ goto out;
>>+
>>+ error = -EINVAL;
>>+ if (new_limits[0] > UB_MAXVALUE || new_limits[1] > UB_MAXVALUE)
>>+ goto out;
>>+
>>+ error = -ENOENT;

>>+ ub = beancounter_findcreate(uid, NULL, 0);
>>+ if (ub == NULL)
>>+ goto out;
>>+
>>+ spin_lock_irqsave(&ub->ub_lock, flags);
>>+ ub->ub_parms[resource].barrier = new_limits[0];
>>+ ub->ub_parms[resource].limit = new_limits[1];
>>+ spin_unlock_irqrestore(&ub->ub_lock, flags);
>>+
>>+ put_beancounter(ub);
>>+ error = 0;
>>+out:
>>+ return error;
>>+}
>>+#endif
>>
>> ---------- ----------
>>Using Tomcat but need to do more? Need to support web services, security?
>>Get stuff done quickly with pre-integrated technology to make your job easier
>>Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
>> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
>>_____
>>ckrm-tech mailing list
>>https://lists.sourceforge.net/lists/listinfo/ckrm-tech
>
>
>

Subject: Re: [ckrm-tech] [PATCH 2/7] UBC: core (structures, API)
Posted by dev on Fri, 18 Aug 2006 11:50:22 GMT
View Forum Message <> Reply to Message

Matt Helsley wrote:
> On Fri, 2006-08-18 at 11:35 +0400, Andrey Savochkin wrote:
>
>>On Thu, Aug 17, 2006 at 10:31:37PM -0700, Andrew Morton wrote:
>>
>>>On Thu, 17 Aug 2006 15:53:40 +0400
>>>Kirill Korotaev <dev@sw.ru> wrote:
>>>
>>>
>>>>>>+struct user_beancounter
>>>>>>+{
>>>>>>+ atomic_t ub_refcount;
>>>>>>+ spinlock_t ub_lock;
>>>>>>+ uid_t   ub_uid;
>>>>>

>>>>>
>>>>>Why uid? Will it be possible to club processes belonging to different
>>>>>users to same bean counter.
>>>>
>>>>oh, its a misname. Should be ub_id. it is ID of user_beancounter
>>>>and has nothing to do with user id.
>>>
>>>But it uses a uid_t. That's more than a misnaming?
>>
>>It used to be uid-related in ancient times when the notion of container
>>hadn't formed up.
>>"user" part of user_beancounter name has the same origin :)
>
>
> Is it similarly irrelevant now? If so perhaps a big rename could be used
> to make the names clearer (s/user_//, s/ub_/bc_/, ...).
hm... let's try :)

Kirill

Subject: Re: [ckrm-tech] [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by dev on Fri, 18 Aug 2006 12:34:23 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Thu, 2006-08-17 at 15:45 +0400, Kirill Korotaev wrote:
>
>>We need more complex decrement/locking scheme than krefs
>>provide. e.g. in __put_beancounter() we need
>>atomic_dec_and_lock_irqsave() semantics for performance optimizations.
>
>
> Is it possible to put the locking in the destructor? It seems like that
> should give similar behavior.
objects live in hashes also so you need to distinguish objects being freed
on lookup somehow.

Kirill

Subject: Re: [ckrm-tech] [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Dave Hansen on Fri, 18 Aug 2006 14:43:08 GMT
View Forum Message <> Reply to Message

On Fri, 2006-08-18 at 12:12 +0400, Kirill Korotaev wrote:
> LDT takes from 1 to 16 pages. and is allocated by vmalloc.

> do you propose to replace it with slab which can fail due to memory
> fragmentation?

Nope. ;)

> the same applies to fdset, fdarray, ipc ids and iptables entries.

The vmalloc area, along with all of those other structures _have_ other
data structures. Now, it will take a wee bit more patching to directly
tag those thing with explicit container pointers (or accounting
references), but I would much prefer that, especially for the things
that are larger than a page.

I worry that this approach was used instead of patching all of the
individual subsystems because this was easier to maintain as an
out-of-tree patch, and it isn't necessarily the best approach.

-- Dave

Subject: Re: [ckrm-tech] [PATCH 4/7] UBC: syscalls (user interface)
Posted by Dave Hansen on Fri, 18 Aug 2006 14:45:48 GMT
View Forum Message <> Reply to Message

On Fri, 2006-08-18 at 12:08 +0400, Andrey Savochkin wrote:
>
> 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.

Hold on here for just a sec...

It is quite possible to do memory management aimed at one container
while that container's memory still participates in the main VM.

There is overhead here, as the LRU scanning mechanisms get less
efficient, but I'd rather pay a penalty at LRU scanning time than divide
up the VM, or coarsely start failing allocations.

-- Dave

Subject: Re: [ckrm-tech] [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Dave Hansen on Fri, 18 Aug 2006 14:58:33 GMT
View Forum Message <> Reply to Message

On Fri, 2006-08-18 at 13:31 +0400, Kirill Korotaev wrote:
> they all are troublesome :/
> user can create lots of vmas, w/o page tables.
> lots of fdsets, ipcids.
> These are not reclaimable.

I guess one of my big questions surrounding these patches is why the accounting is done with pages. If there really is a need to limit these different kernel objects, then why not simply write patches to limit *these* *objects*? I trust there is a very good technical reason for doing this, I just don't understand why, yet.

-- Dave

Subject: Re: [ckrm-tech] [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Dave Hansen on Fri, 18 Aug 2006 15:06:22 GMT
View Forum Message <> Reply to Message

On Fri, 2006-08-18 at 13:31 +0400, Kirill Korotaev wrote:
> they all are troublesome :/
> user can create lots of vmas, w/o page tables.
> lots of fdsets, ipcids.
> These are not reclaimable.

In the real world, with the customers to which you've given these patches, which of these objects is most likely to be consuming the most space? Is there one set of objects that we could work on that would fix _most_ of the cases which you have encountered?

> Also consider the following scenario with reclaimable page tables.
> e.g. user hit kmemsize limit due to fat page tables.
> kernel reclaims some of the page tables and frees user kenerl memory.
> after that user creates some uncreclaimable objects like fdsets or ipcs
> and then accesses memory with reclaimed page tables.
> Sooner or later we kill user with SIGSEGV from page fault due to
> no memory. This is worse then returning ENOMEM from poll() or
> mmap() where user allocates kernel objects.

I think you're claiming that doing reclaim of kernel objects causes much more severe and less recoverable errors than does reclaiming of user pages. That might generally be true, but I have one example that's killing me. (You shouldn't have mentioned mmap ;)

Let's say you have a memory area mapped by one pagetable page, and with 1 user page mapped in it. The system is out of memory, and if you reclaim either the pagetable page or the user page, you're never going to get it back.

So, you pick the user page to reclaim. The user touches it, the memory allocation fails, and the process gets killed.

Instead, you reclaim the pagetable page. The user touches their page, the memory allocation for the pagetable fails, and the process gets killed.

Seems like the same end result to me.

-- Dave

Subject: Re: [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Alan Cox on Fri, 18 Aug 2006 15:39:46 GMT
View Forum Message <> Reply to Message

Ar Iau, 2006-08-17 am 22:31 -0700, ysgrifennodd Andrew Morton:
> > oh, its a misname. Should be ub_id. it is ID of user_beancounter
> > and has nothing to do with user id.
>
> But it uses a uid_t. That's more than a misnaming?

A container id in UBC is an luid which is a type of uid, and uid_t. That follows setluid() in other operating system environments.

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Rohit Seth on Fri, 18 Aug 2006 16:55:58 GMT
View Forum Message <> Reply to Message

On Fri, 2006-08-18 at 13:38 +0400, Kirill Korotaev wrote:
> Rohit Seth wrote:
> > On Thu, 2006-08-17 at 17:27 +0400, Kirill Korotaev wrote:
> >
> >>>If I'm reading this patch right then seems like you are making page
> >>>allocations to fail w/o (for example) trying to purge some pages from
> >>>the page cache belonging to this container. Or is that reclaim going to
> >>>come later?
> >>
> >>charged kernel objects can't be _reclaimed_. how do you propose
> >>to reclaim tasks page tables or files or task struct or vma or etc.?
> >
> >
> >
> > I agree that kernel objects cann't be reclaimed easily. But what you
> > are proposing is also not right. Returning failure w/o doing any

> > reclaim on pages (that are reclaimable) is not useful. And this is why
> > I asked, is this change going to be part of next set of patches (as
> > current set of patches are only tracking kernel usage).

> 1. reclaiming user resources is not that good idea as it looks to you.
> such solutions end up with lots of resources spent on reclaim.
> for user memory reclaims mean consumption of expensive disk I/O bandwidth
> which reduces overall system throughput and influences other users.
>

May be I'm overlooking something very obvious. Please tell me, what happens when a user hits a page fault and the page allocator is easily able to give a page from its pcp list. But container is over its limit of physical memory. In your patch there is no attempt by container support to see if some of the user pages are easily reclaimable. What options a user will have to make sure some room is created.

> 2. kernel memory is mostly not reclaimable. can you reclaim vma structs or ipc ids?

I'm not arguing about that at all. If people want to talk about reclaiming kernel pages then that should be done independent of this subject.

-rohit

Subject: Re: [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Rohit Seth on Fri, 18 Aug 2006 17:51:10 GMT
View Forum Message <> Reply to Message

On Fri, 2006-08-18 at 15:14 +0400, Kirill Korotaev wrote:

> >>2. if you think a bit more about it, adding UB parameters doesn't
> >> require user space changes as well.
> >>3. it is possible to add any kind of interface for UBC. but do you like the idea
> >> to grep 200(containers)x20(parameters) files for getting current usages?
> >
> >
> > How are you doing it currently and how much more efficient it is in
> > comparison to configfs?
> currently it is done with a single file read.
> you can grep it, sum up resources or do what ever you want from bash.
> what is important! you can check whether container hits its limits
> with a single command, while with configs you would have to look through
> 20 files...
>

I think configfs provides all the required functionality that you
listed. You can define the attributes in a such a away that it prints
all the information that you need in one single read operation (I think
the limit is PAGE_SIZE....which is kind of sad).

I've just started playing with configfs for a container implementation
that I'm trying to get a better idea of details.

> IMHO it is convinient to have a text file representing the whole information state
> and system call for applications.
>

There should be an easy interface for shell to be able to do the needful
as well, for example, set the limits.

-rohit

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Chandra Seetharaman on Fri, 18 Aug 2006 19:39:52 GMT
View Forum Message <> Reply to Message

Kirill,

Here are some concerns I have (as of now) w.r.t using UBC for resource
management (in the context of resource groups).

- guarantee support is missing. I do not see any code to provide the
  minimum amount of resource a group can get. It is important for
  providing QoS. (In a different email you did mention guarantee, i am
  referring it here for completeness).
- Creation of a UBC and assignment of task to a UBC always happen in
  the context of the task that is affected. I can understand it works in
  OpenVZ environment, but IMO has issues if one wants it to be used for
  basic resource management
  - application needs to be changed to use this feature.
  - System administrator does not have the control to assign tasks to a
    UBC. Application does by itself.
  - Assignment of task to a UBC need to be transparent to the
    application.
- UBC is deleted when the last task (in that UBC) exits. For resource
  management purposes, UBC should be deleted only when the administrator
  deletes it.
- No ability to set resource specific configuration information.
- No ability to maintain resource specific data in the controller.
- No ability to get the list of tasks belonging to a UBC.

- Doesn't inform the resource controllers when limits(shares) change.
- Doesn't inform the resource controllers when a task's UBC has changed.
- Doesn't recalculate the resource usage when a task's UBC has changed.
  i.e doesn't uncharge the old UBC and charge new UBC.
- For a system administrator name for identification of a UBC is
  better than a number (uid).

regards,

chandra

On Wed, 2006-08-16 at 19:24 +0400, Kirill Korotaev wrote:
> The following patch set presents base of
> User Resource Beancounters (UBC).
> UBC allows to account and control consumption
> of kernel resources used by group of processes.
>
> The full UBC 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. UBC allows 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 UBC and
> management of kernel memory only. Virtual memory
> management will be sent in a couple of days.
>
> The patches in these series are:
> diff-ubc-kconfig.patch:

>   Adds kernel/ub/Kconfig file with UBC options and
>   includes it into arch Kconfigs
>
> diff-ubc-core.patch:
>   Contains core functionality and interfaces of UBC:
>   find/create beancounter, initialization,
>   charge/uncharge of resource, core objects' declarations.
>
> diff-ubc-task.patch:
>   Contains code responsible for setting UB on task,
>   it's inheriting and setting host context in interrupts.
>
>   Task contains three beancounters:
>   1. exec_ub - current context. all resources are charged
>   to this beancounter.
>   2. task_ub - beancounter to which task_struct is charged
>   itself.
>   3. fork_sub - beancounter which is inherited by
>   task's children on fork
>
> diff-ubc-syscalls.patch:
>   Patch adds system calls for UB management:
>   1. sys_getluid   - get current UB id
>   2. sys_setluid   - changes exec_ and fork_ UBs on current
>   3. sys_setublimit - set limits for resources consumtions
>
> diff-ubc-kmem-core.patch:
>   Introduces UB_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_UBC flag - page
>   is charged to current's exec_ub.
>   2. Slabs - kmem_cache may be created with SLAB_UBC flag - in this
>   case each allocation is charged. Caches used by kmalloc are
>   created with SLAB_UBC | SLAB_UBC_NOCHARGE flags. In this case
>   only __GFP_UBC allocations are charged.
>
> diff-ubc-kmem-charge.patch:
>   Adds SLAB_UBC and __GFP_UBC flags in appropriate places
>   to cause charging/limiting of specified resources.
>
> diff-ubc-proc.patch:
>   Adds two proc entries user_beancounters and user_beancounters_sub

> allowing to see current state (usage/limits/fails for each UB).
> Implemented via seq files.
>
> Patch set is applicable to 2.6.18-rc4-mm1
>
> Thanks,
> Kirill
>
>
> ---
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH 1/7] UBC: kconfig
Posted by Chandra Seetharaman on Fri, 18 Aug 2006 19:57:33 GMT
View Forum Message <> Reply to Message

As pointed in an earlier email, it would be better if we could have this
in a arch-independent Kconfig, unless there is any problem with that.

On Wed, 2006-08-16 at 19:35 +0400, Kirill Korotaev wrote:
> Add kernel/ub/Kconfig file with UBC options and
> includes 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/ub/Kconfig | 25 ++++++++++++++++++++++++

> 8 files changed, 39 insertions(+)
>
> --- ./arch/i386/Kconfig.ubkm 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/ub/Kconfig"
> +
> #
> # Use the generic interrupt handling code in kernel/irq/:
> #
> --- ./arch/ia64/Kconfig.ubkm 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/ub/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 "ub/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 "ub/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 "ub/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 "lib/Kconfig"
> --- ./arch/x86_64/Kconfig.ubkm 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/ub/Kconfig"
> --- ./kernel/ub/Kconfig.ubkm 2006-07-28 13:07:38.000000000 +0400
> +++ ./kernel/ub/Kconfig 2006-07-28 13:09:51.000000000 +0400
> @@ -0,0 +1,25 @@
> +#
> +# User resources part (UBC)
> +#
> +# Copyright (C) 2006 OpenVZ. SWsoft Inc
> +
> +menu "User resources"
> +
> +config USER_RESOURCE
> + bool "Enable user resource accounting"
> + default y
> + 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
>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH 3/7] UBC: ub context and inheritance
Posted by Chandra Seetharaman on Fri, 18 Aug 2006 20:03:43 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-16 at 19:38 +0400, Kirill Korotaev wrote:
> Contains code responsible for setting UB on task,
> it's inheriting and setting host context in interrupts.
>
> Task references three beancounters:
> 1. exec_ub current context. all resources are
> charged to this beancounter.
> 2. task_ub beancounter to which task_struct is
> charged itself.

I do not see why task_ub is needed ? i do not see it being used
anywhere.

> 3. fork_sub beancounter which is inherited by
> task's children on fork

>From other emails it looks like renaming fork/exec to be real/effective
will be easier to understand.

>
>
<snip>

--

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

Subject: Re: [ckrm-tech] [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by Chandra Seetharaman on Fri, 18 Aug 2006 20:13:34 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-16 at 19:39 +0400, Kirill Korotaev wrote:

<snip>

> +/*
> + * The setbeanlimit syscall
> + */
> +asmlinkage long sys_setublimit(uid_t uid, unsigned long resource,
> + unsigned long *limits)
> +{
> + int error;
> + unsigned long flags;
> + struct user_beancounter *ub;
> + unsigned long new_limits[2];
> +
> + error = -EPERM;
> + if(!capable(CAP_SYS_RESOURCE))
> + goto out;
> +
> + error = -EINVAL;
> + if (resource >= UB_RESOURCES)
> + goto out;
> +
> + error = -EFAULT;
> + if (copy_from_user(&new_limits, limits, sizeof(new_limits)))
> + goto out;
> +
> + error = -EINVAL;
> + if (new_limits[0] > UB_MAXVALUE || new_limits[1] > UB_MAXVALUE)
> + goto out;
> +
> + error = -ENOENT;
> + ub = beancounter_findcreate(uid, NULL, 0);
> + if (ub == NULL)
> + goto out;
> +
> + spin_lock_irqsave(&ub->ub_lock, flags);
> + ub->ub_parms[resource].barrier = new_limits[0];

> + ub->ub_parms[resource].limit = new_limits[1];

>From my understanding it appear that barrier <= limit. But, the check is
missing here.
> + spin_unlock_irqrestore(&ub->ub_lock, flags);
> +
> + put_beancounter(ub);
> + error = 0;
> +out:
> + return error;
> +}
> +#endif
>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Chandra Seetharaman on Fri, 18 Aug 2006 20:26:19 GMT
View Forum Message <> Reply to Message

Kirill,

IMO, a UBC with resource constraint(limit in this case) should behave no
different than a kernel with limited memory. i.e it should do
reclamation before it starts failing allocation requests. It could even
do it preemptively.

There is no guarantee support which is required for providing QoS.

Each controller modifying the infrastructure code doesn't look good. We
can have proper interfaces to add a new resource controller.

chandra
On Wed, 2006-08-16 at 19:40 +0400, Kirill Korotaev wrote:
> Introduce UB_KMEMSIZE resource which accounts kernel

> objects allocated by task's request.
>
> Reference to UB 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:
> define1. Pages - if allocation is performed with __GFP_UBC flag - page
>   is charged to current's exec_ub.
> 2. Slabs - kmem_cache may be created with SLAB_UBC flag - in this
>   case each allocation is charged. Caches used by kmalloc are
>   created with SLAB_UBC | SLAB_UBC_NOCHARGE flags. In this case
>   only __GFP_UBC allocations are charged.
>
> Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
> Signed-Off-By: Kirill Korotaev <dev@sw.ru>
>
<snip>
--

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

Subject: Re: [RFC][PATCH 1/7] UBC: kconfig
Posted by Adrian Bunk on Fri, 18 Aug 2006 21:14:15 GMT
View Forum Message <> Reply to Message

On Wed, Aug 16, 2006 at 07:35:34PM +0400, Kirill Korotaev wrote:
> Add kernel/ub/Kconfig file with UBC options and
> includes 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/ub/Kconfig | 25 +++++++++++++++++++++++++
> 8 files changed, 39 insertions(+)
>...

> --- ./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 "ub/Kconfig"

kernel/ub/Kconfig

>...
> --- ./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 "ub/Kconfig"

kernel/ub/Kconfig

>...
> --- ./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 "ub/Kconfig"

kernel/ub/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 "lib/Kconfig"

kernel/ub/Kconfig

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

> +#
> +# User resources part (UBC)
> +#
> +# Copyright (C) 2006 OpenVZ. SWsoft Inc
> +
> +menu "User resources"
> +
> +config USER_RESOURCE
> + bool "Enable user resource accounting"
> + default y
>...

Optional functionality shouldn't default to y.

cu
Adrian

--

Gentoo kernels are 42 times more popular than SUSE kernels among
KLive users (a service by SUSE contractor Andrea Arcangeli that
gathers data about kernels from many users worldwide).

There are three kinds of lies: Lies, Damn Lies, and Statistics.
Benjamin Disraeli

Subject: Re: [ckrm-tech] [RFC][PATCH 3/7] UBC: ub context and inheritance
Posted by Matt Helsley on Sat, 19 Aug 2006 02:19:29 GMT
View Forum Message <> Reply to Message

On Fri, 2006-08-18 at 13:23 +0400, Kirill Korotaev wrote:
> Matt Helsley wrote:
> > On Wed, 2006-08-16 at 19:38 +0400, Kirill Korotaev wrote:
> >
> >>Contains code responsible for setting UB on task,
> >>it's inheriting and setting host context in interrupts.
> >>
> >>Task references three beancounters:
> >> 1. exec_ub current context. all resources are
> >> charged to this beancounter.
> >
> >
> > nit: 2-3 below seem to contradict "all". If you mean "the rest" then
> > perhaps you ought to reorder these:
> >
> > 1. task_ub ...
> > 2. fork_sub ...

> > 3. exec_ub Current context. Resources not charged to task_ub
> > or fork_sub are charged to this beancounter.
> not sure what you mean.
> task_ub - where _task_ _itself_ is charged as an object.
> following patches will add charging of "number of tasks" using it.
> fork_sub - beancounter which is inherited on fork() (chaning task beancounter).
> exec_ub - is current context.
>
>
> >> 2. task_ub beancounter to which task_struct is
> >> charged itself.
> > >
> > >
> > > Is task_ub frequently the parent beancounter of exec_ub? If it's always
> > > the parent then perhaps the one or more of these _ub fields in the task
> > > struct are not necessary.
> no, task_ub != exec_ub of parent task
> when task is created anything can happen: task can change ub, parent can change ub,
> task can be reparented. But the UB we charged task to should be known.
>
> > Also in that case keeping copies of the
> > "parent" user_beancounter pointers in the task_beancounters would seem
> > bug-prone -- if the hierarchy of beancounters changes then these would
> > need to be changed too.
> > >
> > >
> > >> 3. fork_sub beancounter which is inherited by
> > >> task's children on fork
> > >
> > >
> > > Is this frequently the same as exec_ub?
> frequently, but not always. exec_ub is changed in softirq for example.
> consider exec_ub as 'current' pointer in kernel.
>
> see other comments below
>
> >>Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
> >>Signed-Off-By: Kirill Korotaev <dev@sw.ru>
> >>
> >>---
> >> include/linux/sched.h | 5 +++++
> >> include/ub/task.h | 42 ++
> >> kernel/fork.c | 21 +++++++++++++++++++-----
> >> kernel/irq/handle.c | 9 +++++++++
> >> kernel/softirq.c | 8 ++++++++
> >> kernel/ub/Makefile | 1 +
> >> kernel/ub/beancounter.c | 4 ++++
> >> kernel/ub/misc.c | 34 ++++++++++++++++++++++++++++++++++

> >> 8 files changed, 119 insertions(+), 5 deletions(-)
> >>
> >>--- ./include/linux/sched.h.ubfork 2006-07-17 17:01:12.000000000 +0400
> >>+++ ./include/linux/sched.h 2006-07-31 16:01:54.000000000 +0400
> >>@@ -81,6 +81,8 @@ struct sched_param {
> >> #include <linux/timer.h>
> >> #include <linux/hrtimer.h>
> >>
> >>+#include <ub/task.h>
> >>+
> >> #include <asm/processor.h>
> >>
> >> struct exec_domain;
> >>@@ -997,6 +999,9 @@ struct task_struct {
> >> spinlock_t delays_lock;
> >> struct task_delay_info *delays;
> >> #endif
> >>+#ifdef CONFIG_USER_RESOURCE
> >>+ struct task_beancounter task_bc;
> >>+#endif
> >> };
> >>
> >> static inline pid_t process_group(struct task_struct *tsk)
> >>--- ./include/ub/task.h.ubfork 2006-07-28 18:53:52.000000000 +0400
> >>+++ ./include/ub/task.h 2006-08-01 15:26:08.000000000 +0400
> >>@@ -0,0 +1,42 @@
> >>+/*
> >>+ * include/ub/task.h
> >>+ *
> >>+ * Copyright (C) 2006 OpenVZ. SWsoft Inc
> >>+ *
> >>+ */
> >>+
> >>+#ifndef __UB_TASK_H_
> >>+#define __UB_TASK_H_
> >>+
> >>+#include <linux/config.h>
> >>+
> >>+struct user_beancounter;
> >>+
> >>+struct task_beancounter {
> >>+ struct user_beancounter *exec_ub;
> >>+ struct user_beancounter *task_ub;
> >>+ struct user_beancounter *fork_sub;
> >>+};
> >>+
> >>+#ifdef CONFIG_USER_RESOURCE
> >>+#define get_exec_ub() (current->task_bc.exec_ub)

> >>+#define set_exec_ub(newub)   \
> >>+ ({   \
> >>+   struct user_beancounter *old; \
> >>+   struct task_beancounter *tbc; \
> >>+   tbc = &current->task_bc; \
> >>+   old = tbc->exec_ub; \
> >>+   tbc->exec_ub = newub; \
> >>+   old;   \
> >>+ })
> >>+
> >
> >
> > How about making these static inlines?
> possible, but this requires including sched.h, which includes this file...
> so this one is easier and more separated.
>
> >>+int ub_task_charge(struct task_struct *parent, struct task_struct *new);
> >>+void ub_task_uncharge(struct task_struct *tsk);
> >>+
> >>+#else /* CONFIG_USER_RESOURCE */
> >>+#define get_exec_ub() (NULL)
> >>+#define set_exec_ub(__ub) (NULL)
> >>+#define ub_task_charge(p, t) (0)
> >>+#define ub_task_uncharge(t) do { } while (0)
> >>+#endif /* CONFIG_USER_RESOURCE */
> >>+#endif /* __UB_TASK_H_ */
> >>--- ./kernel/irq/handle.c.ubirq 2006-07-10 12:39:20.000000000 +0400
> >>+++ ./kernel/irq/handle.c 2006-08-01 12:39:34.000000000 +0400
> >>@@ -16,6 +16,9 @@
> >> #include <linux/interrupt.h>
> >> #include <linux/kernel_stat.h>
> >>
> >>+#include <ub/beancounter.h>
> >>+#include <ub/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 user_beancounter *ub;
> >>+
> >>+ ub = set_exec_ub(&ub0);
> >
> >
> > Perhaps a comment: "/* Don't charge resources gained in interrupts to current */

> ok, will add comment:
> /* UBC charges should be done to host system */
> >
> >
> >> 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);
> >>+
> >>+ (void) set_exec_ub(ub);
> >>   return 1;
> >> }
> >>
> >>@@ -246,6 +254,7 @@ out:
> >> desc->chip->end(irq);
> >> spin_unlock(&desc->lock);
> >>
> >>+ (void) set_exec_ub(ub);
> > >
> > >
> > >
> > > Seems like a WARN_ON() would be appropriate rather than ignoring the
> > > return code.
> > BUG_ON(ret != &ub0) ?

Oops, yes, it's not a return code and BUG_ON() does seem more
appropriate.

>
> maybe introduce a kind of
> reset_exec_ub(old_ub, expected_current_ub)
> {
>   ret = set_exec_ub(old_ub);
>   BUG_ON(ret != expected_current_ub);
> }
> ?

Seems like a good idea to me. This way when UBC is not configured
there'd also be no BUG_ON().

> >> return 1;
> >> }
> >>
> >>--- ./kernel/softirq.c.ubirq 2006-07-17 17:01:12.000000000 +0400
> >>+++ ./kernel/softirq.c 2006-08-01 12:40:44.000000000 +0400
> >>@@ -18,6 +18,9 @@

> >> #include <linux/rcupdate.h>
> >> #include <linux/smp.h>
> >>
> >>+#include <ub/beancounter.h>
> >>+#include <ub/task.h>
> >>+
> >> #include <asm/irq.h>
> >> /*
> >>   - No shared variables, all the data are CPU local.
> >>@@ -191,6 +194,9 @@ asmlinkage void __do_softirq(void)
> >> __u32 pending;
> >> int max_restart = MAX_SOFTIRQ_RESTART;
> >> int cpu;
> >>+ struct user_beancounter *ub;
> >>+
> >>+ ub = set_exec_ub(&ub0);
> >
> >
> > Perhaps add the same comment...
> ok
>
> >
> >
> >> pending = local_softirq_pending();
> >> account_system_vtime(current);
> >>@@ -229,6 +235,8 @@ restart:
> >>
> >> account_system_vtime(current);
> >> _local_bh_enable();
> >>+
> >>+ (void) set_exec_ub(ub);
> >
> >
> > .. and the same WARN_ON.
> >
> >
> >> }
> >>
> >> #ifndef __ARCH_HAS_DO_SOFTIRQ
> >>--- ./kernel/fork.c.ubfork 2006-07-17 17:01:12.000000000 +0400
> >>+++ ./kernel/fork.c 2006-08-01 12:58:36.000000000 +0400
> >>@@ -46,6 +46,8 @@
> >> #include <linux/delayacct.h>
> >> #include <linux/taskstats_kern.h>
> >>
> >>+#include <ub/task.h>
> >>+
> >> #include <asm/pgtable.h>

```
> >> #include <asm/pgalloc.h>
> >> #include <asm/uaccess.h>
> >>@@ -102,6 +104,7 @@ static kmem_cache_t *mm_cachep;
> >>
> >> void free_task(struct task_struct *tsk)
> >> {
> >>+ ub_task_uncharge(tsk);
> >>  free_thread_info(tsk->thread_info);
> >>  rt_mutex_debug_task_free(tsk);
> >>  free_task_struct(tsk);
> >>@@ -162,18 +165,19 @@ static struct task_struct *dup_task_stru
> >>
> >>  tsk = alloc_task_struct();
> >>  if (!tsk)
> >>-  return NULL;
> >>+  goto out;
> >>
> >>  ti = alloc_thread_info(tsk);
> >>- if (!ti) {
> >>-  free_task_struct(tsk);
> >>-  return NULL;
> >>- }
> >>+ if (!ti)
> >>+  goto out_tsk;
> >>
> >>  *tsk = *orig;
> >>  tsk->thread_info = ti;
> >>  setup_thread_stack(tsk, orig);
> >>
> >>+ if (ub_task_charge(orig, tsk))
> >>+  goto out_ti;
> >>+
> >>  /* One for us, one for whoever does the "release_task()" (usually parent) */
> >>  atomic_set(&tsk->usage,2);
> >>  atomic_set(&tsk->fs_excl, 0);
> >>@@ -180,6 +184,13 @@ static struct task_struct *dup_task_stru
> >> #endif
> >>  tsk->splice_pipe = NULL;
> >>  return tsk;
> >>+
> >>+out_ti:
> >>+ free_thread_info(ti);
> >>+out_tsk:
> >>+ free_task_struct(tsk);
> >>+out:
> >>+ return NULL;
> >
> >
```

> > Ugh. This is starting to look like copy_process(). Any reason you
> > couldn't move the bean counter bits to copy_process() instead?
> This is more logical place since we _will_ charge task here
> (next patchset for numproc).
> It is logically better to charge objects in places where
> they are allocated. At the same time we inherit tasks ubs here.

Most other systems that aren't so critically related to task struct
allocation and copying place their code in copy_process() and not in
dup_task_struct().

Frankly this still seems to belong in copy_process(). The pattern (with
the gotos) is already there, as are accounting, audit, and security
pieces for example.

> >> }
> >>
> >> #ifdef CONFIG_MMU
> >>--- ./kernel/ub/Makefile.ubcore 2006-08-03 16:24:56.000000000 +0400
> >>+++ ./kernel/ub/Makefile 2006-08-01 11:08:39.000000000 +0400
> >>@@ -5,3 +5,4 @@
> >> #
> >>
> >> obj-$(CONFIG_USER_RESOURCE) += beancounter.o
> >>+obj-$(CONFIG_USER_RESOURCE) += misc.o
> >>--- ./kernel/ub/beancounter.c.ubcore 2006-07-28 13:07:44.000000000 +0400
> >>+++ ./kernel/ub/beancounter.c 2006-08-03 16:14:17.000000000 +0400
> >>@@ -395,6 +395,10 @@
> >> spin_lock_init(&ub_hash_lock);
> >> slot = &ub_hash[ub_hash_fun(ub->ub_uid)];
> >> hlist_add_head(&ub->hash, slot);
> >>+
> >>+ current->task_bc.exec_ub = ub;
> >>+ current->task_bc.task_ub = get_beancounter(ub);
> >>+ current->task_bc.fork_sub = get_beancounter(ub);
> >> }
> >>
> >> void __init ub_init_late(void)
> >>--- ./kernel/ub/misc.c.ubfork 2006-07-31 16:23:44.000000000 +0400
> >>+++ ./kernel/ub/misc.c 2006-07-31 16:28:47.000000000 +0400
> >>@@ -0,0 +1,34 @@
> >>+/*
> >>+ * kernel/ub/misc.c
> >>+ *
> >>+ * Copyright (C) 2006 OpenVZ. SWsoft Inc.
> >>+ *
> >>+ */
> >>+

> >>+#include <linux/sched.h>
> >>+
> >>+#include <ub/beancounter.h>
> >>+#include <ub/task.h>
> >>+
> >>+int ub_task_charge(struct task_struct *parent, struct task_struct *new)
> >>+{
> >
> >
> > parent could be derived from new if you move the charge to copy_process
> > instead of dup_task_struct.
> we can split it into:
> ub_charge_task() in dup_task_struct to account _task_ itself.
> ub_copy_process() in copy_process() to inherit and initialize
>   exec_ub and fork_sub
>
> what do you think?

 I do like the idea of splitting this up.

 Though as I said I'm still against adding stuff to dup_task_struct() if it's not allocating/copying the task struct or thread info. I think of "dup_task_struct()" as dealing with the core purpose of the task_struct -- copy_process() seems to be for all of the things that have plugged their own fields into task_struct over the years.

 Since I haven't seen the numproc patches that follow this I can't really comment (one way or the other) on whether plugging that into dup_task_struct() seems appropriate or necessary.

> >>+ struct task_beancounter *old_bc;
> >>+ struct task_beancounter *new_bc;
> >>+ struct user_beancounter *ub;
> >>+
> >>+ old_bc = &parent->task_bc;
> >>+ new_bc = &new->task_bc;
> >>+
> >>+ ub = old_bc->fork_sub;
> >>+ new_bc->exec_ub = get_beancounter(ub);
> >>+ new_bc->task_ub = get_beancounter(ub);
> >>+ new_bc->fork_sub = get_beancounter(ub);
> >>+ return 0;
> >>+}

<snip>

Cheers,
 -Matt Helsley

Subject: Re: [ckrm-tech] [RFC][PATCH 4/7] UBC: syscalls (user interface)
Posted by Matt Helsley on Sat, 19 Aug 2006 02:43:02 GMT
View Forum Message <> Reply to Message

On Fri, 2006-08-18 at 15:45 +0400, Kirill Korotaev wrote:
> Matt Helsley wrote:
>
> [... snip ...]
> >>--- ./kernel/ub/sys.c.ubsys 2006-07-28 18:52:18.000000000 +0400
> >>+++ ./kernel/ub/sys.c 2006-08-03 16:14:23.000000000 +0400
> >>@@ -0,0 +1,126 @@

<snip>

> >>+#else /* CONFIG_USER_RESOURCE */
> >>+
> >>+/*
> >>+ * The (rather boring) getluid syscall
> >>+ */
> >>+asmlinkage long sys_getluid(void)
> >>+{
> >>+ struct user_beancounter *ub;
> >>+
> >>+ ub = get_exec_ub();
> >>+ if (ub == NULL)
> >>+ return -EINVAL;
> >>+
> >>+ return ub->ub_uid;
> >>+}
> >>+
> >>+/*
> >>+ * The setluid syscall
> >>+ */
> >>+asmlinkage long sys_setluid(uid_t uid)
> >>+{
> >>+ int error;
> >>+ struct user_beancounter *ub;
> >>+ struct task_beancounter *task_bc;
> >>+
> >>+ task_bc = &current->task_bc;
> >>+
> >>+ /* You may not disown a setluid */
> >>+ error = -EINVAL;
> >>+ if (uid == (uid_t)-1)
> >>+ goto out;
> >>+
> >>+ /* You may only set an ub as root */
> >>+ error = -EPERM;
> >>+ if (!capable(CAP_SETUID))

> >>+ goto out;
> >
> >
> > With resource groups you don't necessarily have to be root -- just the
> > owner of the group and task.
> the question is - who is the owner of group?

Whoever is made the 'owner' of the directory is the owner of the group.
If you own both then you can add your task to your group.

> user, user group or who?
> Both are bad, since the same user can run inside the container and thus
> container will be potentially controllable/breakable from inside.

 No, that's not a problem. The way shares work is you get a "portion" of
the parent group's resources and if the parent has limited your portion
you cannot exceed that. At the same time you can control how your
portion is dealt out within the child group.

> > Filesystems and appropriate share representations offer a way to give
> > regular users the ability to manage their resources without requiring
> > CAP_FOO.
> not sure what you propose...

A filesystem interface.

> we can introduce the following rules:
>
> containers (UB) can be created by process with SETUID cap only.
> subcontainers (SUB) can be created by any process.

Can subsubcontainers be created?

> what do you think?

I think a filesystem interface would work better. ;)

>
> >>+ /* Ok - set up a beancounter entry for this user */
> >>+ error = -ENOBUFS;
> >>+ ub = beancounter_findcreate(uid, NULL, UB_ALLOC);
> >>+ if (ub == NULL)
> >>+ goto out;
> >>+
> >>+ /* install bc */
> >>+ put_beancounter(task_bc->exec_ub);
> >>+ task_bc->exec_ub = ub;
> >>+ put_beancounter(task_bc->fork_sub);

> >>+ task_bc->fork_sub = get_beancounter(ub);
> >>+ error = 0;
> >>+out:
> >>+ return error;
> >>+}
> >>+
> >>+/*
> >>+ * The setbeanlimit syscall
> >>+ */
> >>+asmlinkage long sys_setublimit(uid_t uid, unsigned long resource,
> >>+ unsigned long *limits)
> >>+{
> >>+ int error;
> >>+ unsigned long flags;
> >>+ struct user_beancounter *ub;
> >>+ unsigned long new_limits[2];
> >>+
> >>+ error = -EPERM;
> >>+ if(!capable(CAP_SYS_RESOURCE))
> >>+ goto out;
> >
> >
> > Again, a filesystem interface would give us more flexibility when it
> > comes to allowing users to manage their resources while still preventing
> > them from exceeding limits.
> we can have 2 different root users with uid = 0 in 2 different containers.

 You shouldn't need to have the 2 containers to give resource control to
other users. In other words you shouldn't need to use containers in
order to do resource management. The container model is by no means the
only way to model resource management.

> > I doubt you really want to give owners of a container CAP_SYS_RESOURCE
> > and CAP_USER (i.e. total control over resource management) just to allow
> > them to manage their subset of the resources.
> The origin idea is that administator of the node can manage user
> resources only. Users can't, since otherwise they can increase the limits.

 The user may wish to manage the resource usage of her applications
within restrictions imposed by an administrator. If the user has a
portion of resources then you only need to ensure that the sum of her
resources does not exceed the administrator-provided limit.

> But we can allow them to manage sub beancoutners imho...

And subsubbeancounters?

<snip>

Cheers,
 -Matt Helsley

Subject: Re: [ckrm-tech] [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Balbir Singh on Sun, 20 Aug 2006 04:58:41 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:

> +/*
> + * Resource list.
> + */
> +
> +#define UB_RESOURCES 0
> +
> +struct ubparm {
> + /*
> + * 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 barrier;
> + /* hard resource limit */
> + unsigned long limit;
> + /* consumed resources */
> + unsigned long held;
> + /* maximum amount of consumed resources through the last period */
> + unsigned long maxheld;
> + /* minimum amount of consumed resources through the last period */
> + unsigned long minheld;
> + /* count of failed charges */
> + unsigned long failcnt;
> +};
> +

Comments to the side of the field would make it easier to read and understand
the structure. I think there are already other comments requesting for renaming
of the barrier field to hard_limit.

<snip>

> +static inline void ub_adjust_held_minmax(struct user_beancounter *ub,
> + int resource)
> +{
> + if (ub->ub_parms[resource].maxheld < ub->ub_parms[resource].held)
> + ub->ub_parms[resource].maxheld = ub->ub_parms[resource].held;

> + if (ub->ub_parms[resource].minheld > ub->ub_parms[resource].held)
> + ub->ub_parms[resource].minheld = ub->ub_parms[resource].held;
> +}

A comment here to clarify what the function does would be helpful, specially due
to the comparison above

if (maxheld < held)
 maxheld = held
if (minheld > held)
 minheld = held

<snip>

> +struct user_beancounter ub0;

How about global_ub or init_ub?

> +
> +#define ub_hash_fun(x) ((((x) >> 8) ^ (x)) & (UB_HASH_SIZE - 1))
> +#define ub_subhash_fun(p, id) ub_hash_fun((p)->ub_uid + (id) * 17)
> +

What hash properties are we looking for in the hash function? Is the hash
function universal?

> +struct hlist_head ub_hash[UB_HASH_SIZE];
> +spinlock_t ub_hash_lock;
> +
> +EXPORT_SYMBOL(ub_hash);
> +EXPORT_SYMBOL(ub_hash_lock);
> +
> +/*
> + * Per user 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 user_beancounter *beancounter_findcreate(uid_t uid,
> + struct user_beancounter *p, int mask)
> +{
> + struct user_beancounter *new_ub, *ub, *tmpl_ub;

```
> + unsigned long flags;
> + struct hlist_head *slot;
> + struct hlist_node *pos;
> +
> + if (mask & UB_LOOKUP_SUB) {
> +  WARN_ON(p == NULL);
> +  tmpl_ub = &default_subbeancounter;
> +  slot = &ub_hash[ub_subhash_fun(p, uid)];
> + } else {
> +  WARN_ON(p != NULL);
> +  tmpl_ub = &default_beancounter;
> +  slot = &ub_hash[ub_hash_fun(uid)];
> + }
> + new_ub = NULL;
> +
> +retry:
> + spin_lock_irqsave(&ub_hash_lock, flags);
> + hlist_for_each_entry (ub, pos, slot, hash)
> +  if (ub->ub_uid == uid && ub->parent == p)
> +   break;
> +
> + if (pos != NULL) {
> +  get_beancounter(ub);
> +  spin_unlock_irqrestore(&ub_hash_lock, flags);
> +
> +  if (new_ub != NULL) {
> +   put_beancounter(new_ub->parent);
> +   kmem_cache_free(ub_cachep, new_ub);
> +  }
```

A comment indicative of this being a part of race handing would be useful.
Could you please consider refactoring this function if possible.

```
> +  return ub;
> + }
> +
> + if (!(mask & UB_ALLOC))
> +  goto out_unlock;
> +
> + if (new_ub != NULL)
> +  goto out_install;
> +
> + if (mask & UB_ALLOC_ATOMIC) {
> +  new_ub = kmem_cache_alloc(ub_cachep, GFP_ATOMIC);
> +  if (new_ub == NULL)
> +   goto out_unlock;
> +
> +  memcpy(new_ub, tmpl_ub, sizeof(*new_ub));
```

```
> +  init_beancounter_struct(new_ub, uid);
> +  if (p)
> +   new_ub->parent = get_beancounter(p);
> +  goto out_install;
> + }
> +
> + spin_unlock_irqrestore(&ub_hash_lock, flags);
> +
> + new_ub = kmem_cache_alloc(ub_cachep, GFP_KERNEL);
> + if (new_ub == NULL)
> +  goto out;
> +
> + memcpy(new_ub, tmpl_ub, sizeof(*new_ub));
> + init_beancounter_struct(new_ub, uid);
> + if (p)
> +  new_ub->parent = get_beancounter(p);
> + goto retry;
> +
> +out_install:
> + hlist_add_head(&new_ub->hash, slot);
> +out_unlock:
> + spin_unlock_irqrestore(&ub_hash_lock, flags);
> +out:
> + return new_ub;
> +}
> +
> +EXPORT_SYMBOL(beancounter_findcreate);
> +
```

<snip>

```
> +void __put_beancounter(struct user_beancounter *ub)
> +{
> + unsigned long flags;
> + struct user_beancounter *parent;
> +
> +again:
> + parent = ub->parent;
> + /* equevalent to atomic_dec_and_lock_irqsave() */
> + local_irq_save(flags);
> + if (likely(!atomic_dec_and_lock(&ub->ub_refcount, &ub_hash_lock))) {
> +  if (unlikely(atomic_read(&ub->ub_refcount) < 0))
> +   printk(KERN_ERR "UB: Bad ub refcount: ub=%p, "
> +     "luid=%d, ref=%d\n",
> +     ub, ub->ub_uid,
> +     atomic_read(&ub->ub_refcount));
> +  local_irq_restore(flags);
> +  return;
```

> + }
> +
> + if (unlikely(ub == &ub0)) {
> + printk(KERN_ERR "Trying to put ub0\n");
> + spin_unlock_irqrestore(&ub_hash_lock, flags);
> + return;
> + }
> +
> + verify_held(ub);
> + hlist_del(&ub->hash);
> + spin_unlock_irqrestore(&ub_hash_lock, flags);

Is this function called with the ub_hash_lock held()? A comment would be useful
or you could call it __put_beancounter_locked :-)

> +
> + kmem_cache_free(ub_cachep, ub);
> +
> + ub = parent;
> + if (ub != NULL)
> + goto again;

Could you please convert this to a do {} while() loop.

> +}
> +
> +EXPORT_SYMBOL(__put_beancounter);

<snip>

> +int charge_beancounter(struct user_beancounter *ub,
> + int resource, unsigned long val, enum severity strict)
> +{
> + int retval;
> + struct user_beancounter *p, *q;
> + unsigned long flags;
> +
> + retval = -EINVAL;
> + BUG_ON(val > UB_MAXVALUE);
> +
> + local_irq_save(flags);
> + for (p = ub; p != NULL; p = p->parent) {
> + spin_lock(&p->ub_lock);
> + retval = __charge_beancounter_locked(p, resource, val, strict);

Everyone in the hierarchy is charged the same amount - val?

> + spin_unlock(&p->ub_lock);

> + if (retval)
> +   goto unroll;
> + }
> +out_restore:
> + local_irq_restore(flags);
> + return retval;
> +
> +unroll:
> + for (q = ub; q != p; q = q->parent) {
> + spin_lock(&q->ub_lock);
> + __uncharge_beancounter_locked(q, resource, val);
> + spin_unlock(&q->ub_lock);
> + }
> + goto out_restore;

Too many goto's in both directions - please consider refactoring

> +void charge_beancounter_notop(struct user_beancounter *ub,
> + int resource, unsigned long val)

Whats the meaning of notop?

> +{
> + struct user_beancounter *p;
> + unsigned long flags;
> +
> + local_irq_save(flags);
> + for (p = ub; p->parent != NULL; p = p->parent) {
> + spin_lock(&p->ub_lock);
> + __charge_beancounter_locked(p, resource, val, UB_FORCE);
> + spin_unlock(&p->ub_lock);
> + }
> + local_irq_restore(flags);
> +}
> +

Could some of this code be shared with charge_beancounter to avoid duplication?

> +EXPORT_SYMBOL(charge_beancounter_notop);
> +
> +void __uncharge_beancounter_locked(struct user_beancounter *ub,
> + int resource, unsigned long val)
> +{
> + if (unlikely(ub->ub_parms[resource].held < val)) {
> + ub_print_resource_warning(ub, resource,
> +   "uncharging too much", val, 0);
> + val = ub->ub_parms[resource].held;
> + }

```
> + ub->ub_parms[resource].held -= val;
> + ub_adjust_held_minmax(ub, resource);
> +}
> +
> +void uncharge_beancounter(struct user_beancounter *ub,
> +  int resource, unsigned long val)
> +{
> + unsigned long flags;
> + struct user_beancounter *p;
> +
> + for (p = ub; p != NULL; p = p->parent) {
> +  spin_lock_irqsave(&p->ub_lock, flags);
> +  __uncharge_beancounter_locked(p, resource, val);
> +  spin_unlock_irqrestore(&p->ub_lock, flags);
> + }
> +}
> +
> +EXPORT_SYMBOL(uncharge_beancounter);
> +
> +void uncharge_beancounter_notop(struct user_beancounter *ub,
> +  int resource, unsigned long val)
> +{
> + struct user_beancounter *p;
> + unsigned long flags;
> +
> + local_irq_save(flags);
> + for (p = ub; p->parent != NULL; p = p->parent) {
> +  spin_lock(&p->ub_lock);
> +  __uncharge_beancounter_locked(p, resource, val);
> +  spin_unlock(&p->ub_lock);
> + }
> + local_irq_restore(flags);
> +}
> +
```

The code for both uncharge_beancounter() and uncharge_beancounter_notop() seems to do the same thing

```
> +
> +void __init ub_init_late(void)
> +{
> + struct user_beancounter *ub;
> +
> + ub_cachep = kmem_cache_create("user_beancounters",
> +   sizeof(struct user_beancounter),
> +   0, SLAB_HWCACHE_ALIGN, NULL, NULL);
> + if (ub_cachep == NULL)
> +  panic("Can't create ubc caches\n");
```

> +
> + ub = &default_beancounter;

Whats the relationship between ub0 and default_beancounter?

> + memset(ub, 0, sizeof(default_beancounter));
> + init_beancounter_syslimits(ub);
> + init_beancounter_struct(ub, 0);

Do we need to memset static global variables to 0?
> +
> + ub = &default_subbeancounter;
> + memset(ub, 0, sizeof(default_subbeancounter));
> + init_beancounter_nolimits(ub);
> + init_beancounter_struct(ub, 0);

Do we need to memset static global variables to 0?

> +}
>
> ---
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech

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

Subject: Re: [ckrm-tech] [RFC][PATCH 2/7] UBC: core (structures, API)
Posted by Balbir Singh on Sun, 20 Aug 2006 05:01:02 GMT
View Forum Message <> Reply to Message

> +
> +void __put_beancounter(struct user_beancounter *ub)
> +{
> + unsigned long flags;
> + struct user_beancounter *parent;
> +
> +again:

```
> + parent = ub->parent;
> + /* equevalent to atomic_dec_and_lock_irqsave() */
> + local_irq_save(flags);
> + if (likely(!atomic_dec_and_lock(&ub->ub_refcount, &ub_hash_lock))) {
> +  if (unlikely(atomic_read(&ub->ub_refcount) < 0))
> +   printk(KERN_ERR "UB: Bad ub refcount: ub=%p, "
> +     "luid=%d, ref=%d\n",
> +     ub, ub->ub_uid,
> +     atomic_read(&ub->ub_refcount));
> +  local_irq_restore(flags);
> +  return;
```

Minor comment - the printk (I think there is one other place) could come after the local_irq_restore()

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH 4/7] UBC: syscalls (user interface)
Posted by Magnus Damm on Mon, 21 Aug 2006 02:47:06 GMT
View Forum Message <> Reply to Message

On Fri, 2006-08-18 at 07:45 -0700, Dave Hansen wrote:
> On Fri, 2006-08-18 at 12:08 +0400, Andrey Savochkin wrote:
> >
> > 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.
>
> Hold on here for just a sec...
>
> It is quite possible to do memory management aimed at one container
> while that container's memory still participates in the main VM.
>
> There is overhead here, as the LRU scanning mechanisms get less
> efficient, but I'd rather pay a penalty at LRU scanning time than divide
> up the VM, or coarsely start failing allocations.

This could of course be solved with one LRU per container, which is how the CKRM memory controller implemented things about a year ago.

/ magnus

Subject: Re: [ckrm-tech] [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by dev on Mon, 21 Aug 2006 08:56:08 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Fri, 2006-08-18 at 12:12 +0400, Kirill Korotaev wrote:
>
>>LDT takes from 1 to 16 pages. and is allocated by vmalloc.
>>do you propose to replace it with slab which can fail due to memory
>>fragmentation?
>
>
> Nope. ;)
so what is your proposal then? Sorry, probably missed it due to lots of emails :)

>>the same applies to fdset, fdarray, ipc ids and iptables entries.
>
>
> The vmalloc area, along with all of those other structures _have_ other
> data structures. Now, it will take a wee bit more patching to directly
> tag those thing with explicit container pointers (or accounting
> references), but I would much prefer that, especially for the things
> that are larger than a page.
do you mean that you prefer adding a explicit pointer to the structures
itself?

> I worry that this approach was used instead of patching all of the
> individual subsystems because this was easier to maintain as an
> out-of-tree patch, and it isn't necessarily the best approach.
:) if we were to optimize for patch size then we would select vserver
approach and be happy...

Dave, we used to add UBC pointers on each data structure and then do
a separate accounting in the places where objects are allocated.
We spent a lot of time and investigation on how to make it better,
because it was leading to often accounting errors, wrong error paths etc.
The approach provided in this patchset proved to be much more efficient
and more error prone. And it is much much more elegant!

Thanks,
Kirill

Subject: Re: [ckrm-tech] [RFC][PATCH 3/7] UBC: ub context and inheritance
Posted by dev on Mon, 21 Aug 2006 10:30:29 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:

> On Wed, 2006-08-16 at 19:38 +0400, Kirill Korotaev wrote:
>
>>Contains code responsible for setting UB on task,
>>it's inheriting and setting host context in interrupts.
>>
>>Task references three beancounters:
>> 1. exec_ub current context. all resources are
>>   charged to this beancounter.
>> 2. task_ub beancounter to which task_struct is
>>   charged itself.
>
>
> I do not see why task_ub is needed ? i do not see it being used
> anywhere.
it is used to charge task itself. will be heavily used in next patch set
adding "numproc" UBC parameter.

>> 3. fork_sub beancounter which is inherited by
>>   task's children on fork
>
>
>>From other emails it looks like renaming fork/exec to be real/effective
> will be easier to understand.
there is no "real". exec_ub is effective indeed,
but fork_sub is the one to inherit on fork().

Kirill

Subject: Re: [ckrm-tech] [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by dev on Mon, 21 Aug 2006 10:38:17 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Fri, 2006-08-18 at 13:31 +0400, Kirill Korotaev wrote:
>
>>they all are troublesome :/
>>user can create lots of vmas, w/o page tables.
>>lots of fdsets, ipcids.
>>These are not reclaimable.
>
>
> I guess one of my big questions surrounding these patches is why the
> accounting is done with pages.
probably you missed patch details a bit.
accounting is done:
1. in pages for objects allocated by buddy allocator
2. in slabs for objects allocated from caches

> If there really is a need to limit these
> different kernel objects, then why not simply write patches to limit
> *these* *objects*? I trust there is a very good technical reason for
> doing this, I just don't understand why, yet.
The one reason is that such an accounting allows to estimate the memory
used/required by containers, while limitations by objects:
- per object accounting/limitations do not provide any memory estimation
- having a big number of reasonably high limits still allows the user
  to consume big amount of memory. I.e. the sum of all the limits tend
  to be high and potentially DoS exploitable :/
- memory is easier to setup/control from user POV.
  having hundreds of controls is good, but not much user friendly.

Thanks,
Kirill

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by dev on Mon, 21 Aug 2006 10:41:28 GMT
View Forum Message <> Reply to Message

>>1. reclaiming user resources is not that good idea as it looks to you.
>>such solutions end up with lots of resources spent on reclaim.
>>for user memory reclaims mean consumption of expensive disk I/O bandwidth
>>which reduces overall system throughput and influences other users.
>>
>
>
> May be I'm overlooking something very obvious. Please tell me, what
> happens when a user hits a page fault and the page allocator is easily
> able to give a page from its pcp list. But container is over its limit
> of physical memory. In your patch there is no attempt by container
> support to see if some of the user pages are easily reclaimable. What
> options a user will have to make sure some room is created.
The patch set send doesn't control user memory!
This topic is about kernel memory...

>>2. kernel memory is mostly not reclaimable. can you reclaim vma structs or ipc ids?
>
>
> I'm not arguing about that at all. If people want to talk about
> reclaiming kernel pages then that should be done independent of this
> subject.
Then why do you mess user pages accounting into this thread then?

Kirill

Subject: Re: [ckrm-tech] [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by dev on Mon, 21 Aug 2006 10:48:55 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:
> Kirill,
>
> IMO, a UBC with resource constraint(limit in this case) should behave no
> different than a kernel with limited memory. i.e it should do
> reclamation before it starts failing allocation requests. It could even
> do it preemptively.
first, please notice, that this thread is not about user memory.
we can discuss it later when about to control user memory. And
I still need to notice, that different models of user memory control
can exist. With and without reclamation.

> There is no guarantee support which is required for providing QoS.
where? in UBC? in UBC _there_ are guarentees, even in regard to OOM killer.

> Each controller modifying the infrastructure code doesn't look good. We
> can have proper interfaces to add a new resource controller.
controllers do not modify interfaces nor core. They just add
themself to the list of resources and setup default limits.
do you think it is worth creating infrastructure for these
2 one-line-changes?

> chandra
> On Wed, 2006-08-16 at 19:40 +0400, Kirill Korotaev wrote:
>
>>Introduce UB_KMEMSIZE resource which accounts kernel
>>objects allocated by task's request.
>>
>>Reference to UB 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:
>> define1. Pages - if allocation is performed with __GFP_UBC flag - page
>>   is charged to current's exec_ub.
>> 2. Slabs - kmem_cache may be created with SLAB_UBC flag - in this
>>   case each allocation is charged. Caches used by kmalloc are
>>   created with SLAB_UBC | SLAB_UBC_NOCHARGE flags. In this case
>>   only __GFP_UBC allocations are charged.
>>
>>Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
>>Signed-Off-By: Kirill Korotaev <dev@sw.ru>
>>
> <snip>

Subject: Re: [ckrm-tech] [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by dev on Mon, 21 Aug 2006 12:36:24 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Fri, 2006-08-18 at 13:31 +0400, Kirill Korotaev wrote:
>
>>they all are troublesome :/
>>user can create lots of vmas, w/o page tables.
>>lots of fdsets, ipcids.
>>These are not reclaimable.
>
>
> In the real world, with the customers to which you've given these
> patches, which of these objects is most likely to be consuming the most
> space? Is there one set of objects that we could work on that would fix
> _most_ of the cases which you have encountered?
the question is not about which one consumes more in "real life".
The question is "which of the resources are allocated on user demand
and should be limited for the environment to be secure and isolated".

>>Also consider the following scenario with reclaimable page tables.
>>e.g. user hit kmemsize limit due to fat page tables.
>>kernel reclaims some of the page tables and frees user kenerl memory.
>>after that user creates some uncreclaimable objects like fdsets or ipcs
>>and then accesses memory with reclaimed page tables.
>>Sooner or later we kill user with SIGSEGV from page fault due to
>>no memory. This is worse then returning ENOMEM from poll() or
>>mmap() where user allocates kernel objects.
>
>
> I think you're claiming that doing reclaim of kernel objects causes much
> more severe and less recoverable errors than does reclaiming of user
> pages.
I also claim that reclaiming some of kernel pages is almost undoable :)
Look, the page can be used by slab objects from 2 different containers.
Why one container should suffer from the second one which needs to be reclaimed?
What to do? separate allocators per container? And if you want to reclaim
say a page with vma you need to replaces tons of pointers all over the objects
in SMP safe manner.

That might generally be true, but I have one example that's
> killing me. (You shouldn't have mentioned mmap ;)
>
> Let's say you have a memory area mapped by one pagetable page, and with
> 1 user page mapped in it. The system is out of memory, and if you
> reclaim either the pagetable page or the user page, you're never going
> to get it back.
>

> So, you pick the user page to reclaim. The user touches it, the memory
> allocation fails, and the process gets killed.
>
> Instead, you reclaim the pagetable page. The user touches their page,
> the memory allocation for the pagetable fails, and the process gets
> killed.
>
> Seems like the same end result to me.
because you suggest the same limit for pagetables and user memory.
Thats why we have separate kmemsize limit for kernel objects and privvmpages for
user memory.
privvmpages limit hit will result in -ENOMEM on mmap() system call,
which is memory friendly.

Kirill

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by dev on Mon, 21 Aug 2006 13:21:48 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:
> Kirill,
>
> Here are some concerns I have (as of now) w.r.t using UBC for resource
> management (in the context of resource groups).
>
> - guarantee support is missing. I do not see any code to provide the
>   minimum amount of resource a group can get. It is important for
>   providing QoS. (In a different email you did mention guarantee, i am
>   referring it here for completeness).
I mentioned a couple of times that this is a limited core functionality
in this patch set.
guarantees are implementable as a separate UBC parameters.

> - Creation of a UBC and assignment of task to a UBC always happen in
>   the context of the task that is affected. I can understand it works in
>   OpenVZ environment, but IMO has issues if one wants it to be used for
>   basic resource management
>   - application needs to be changed to use this feature.
>   - System administrator does not have the control to assign tasks to a
>   UBC. Application does by itself.
>   - Assignment of task to a UBC need to be transparent to the
>   application.
this is not 100% true.
UBC itself doesn't prevent from changing context on the fly.
But since this leads to part of resources to be charged to
one UBC and another part to another UBC and so long and so

forth, we believe that more clear and correct interface is
something like fork()/exec()-required-application.

So you can always execute new applications in desired UB and
NO application modification are required.

> - UBC is deleted when the last task (in that UBC) exits. For resource
> management purposes, UBC should be deleted only when the administrator
> deletes it.
1. UBCs are freed when last _resource_ using it puts the last reference.
not the task. And it is a BIG error IMHO to think that resource
management should group tasks. No, it should group _objects_. Tasks
are just the same objects like say sockets.
2. this is easily changeable. You are the only who requested it so far.
3. kernel does so for many other objects like users and no one complains :)

> - No ability to set resource specific configuration information.
UBC model allows to _limit_ users. It is _core_.
We want to do resource management step by step and send it patch by patch,
while you are trying to solve everything at once.

sys_open() for example doesn't allow to open sockets, does it?
the same for UBC. They do what they are supposed to do.

> - No ability to maintain resource specific data in the controller.
it's false. fields can be added to user_beancounter struct easily.
and that's what our controllers do.

> - No ability to get the list of tasks belonging to a UBC.
it is not true. it can be read from /proc or system calls interface,
just like the way one finds all tasks belonging to one user :)

BTW, what is so valueable in this feature?
do you want to have interfaces to find kernel structures and even pages
which belong to the container? tasks are just one type of objects...

> - Doesn't inform the resource controllers when limits(shares) change.
As was answered and noted by Alan Cox:
1. no one defined what type of action should be done when limits change
2. it is extendable _when_ needed. Do you want to introduce hooks just
to have them?
3. is it so BIG obstacle for UBC patch? These 3-lines hooks code which
is not used?

> - Doesn't inform the resource controllers when a task's UBC has changed.
the same as above. we don't add functionality which is not used YET
(and no one even knows HOW).

> - Doesn't recalculate the resource usage when a task's UBC has changed.
> i.e doesn't uncharge the old UBC and charge new UBC.
You probably missed my explanation, that most
resources (except for the simplest one - numproc) can't be recharged
easily. And nothing in UBC code prevents such recharge to be added later
if requested.

> - For a system administrator name for identification of a UBC is
> better than a number (uid).
Have you any problems with pids, uids, gids and signals?
It is a question of interface. I don't mind in changing UBC interface even
to configfs if someone really wants it.

Thanks,
Kirill

Subject: Re: [ckrm-tech] [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Dave Hansen on Mon, 21 Aug 2006 15:10:56 GMT
View Forum Message <> Reply to Message

On Mon, 2006-08-21 at 14:40 +0400, Kirill Korotaev wrote:
> The one reason is that such an accounting allows to estimate the
> memory
> used/required by containers, while limitations by objects:
> - per object accounting/limitations do not provide any memory
> estimation

I know you're more clever than _that_. ;)

> - having a big number of reasonably high limits still allows the user
> to consume big amount of memory. I.e. the sum of all the limits tend
> to be high and potentially DoS exploitable :/
> - memory is easier to setup/control from user POV.
> having hundreds of controls is good, but not much user friendly.

I'm actually starting to think that some accounting memory usage *only*
in the slab, plus a few other structures for any stragglers not using
the slab would suffice. Since the slab knows the size of the objects,
there is no ambiguity about how many are being used. It should also be
a pretty generic way to control individual object types, if anyone else
should ever need it.

The high level pages-used-per-container metric is really nice for just
that, containers. But, I wonder if other users would find it useful if
there were more precise ways of controlling individual object usage.

-- Dave

Subject: Re: [ckrm-tech] [RFC][PATCH 3/7] UBC: ub context and inheritance
Posted by Chandra Seetharaman on Mon, 21 Aug 2006 20:48:21 GMT
View Forum Message <> Reply to Message

On Mon, 2006-08-21 at 14:32 +0400, Kirill Korotaev wrote:
> Chandra Seetharaman wrote:
> > On Wed, 2006-08-16 at 19:38 +0400, Kirill Korotaev wrote:
> >
> >>Contains code responsible for setting UB on task,
> >>it's inheriting and setting host context in interrupts.
> >>
> >>Task references three beancounters:
> >> 1. exec_ub current context. all resources are
> >>   charged to this beancounter.
> >> 2. task_ub beancounter to which task_struct is
> >>   charged itself.
> >
> >
> > I do not see why task_ub is needed ? i do not see it being used
> > anywhere.
> it is used to charge task itself. will be heavily used in next patch set
> adding "numproc" UBC parameter.

Well, from your other responses it sounded like you are including
code/data structure/functionality only when they are used. So, I wasn't
clear if this is missed out on cleanup or really part of UBC.

Besides, if this is needed only for a specific controller, shouldn't the
controller worry about maintaining it ?

>
>> 3. fork_sub beancounter which is inherited by
>>   task's children on fork
> >
> >
>>From other emails it looks like renaming fork/exec to be real/effective
> > will be easier to understand.
> there is no "real". exec_ub is effective indeed,
> but fork_sub is the one to inherit on fork().

IMO, fork_cb/exec_cb doesn't convey the real meaning of the usage.
>
> Kirill
>
>

> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Chandra Seetharaman on Mon, 21 Aug 2006 20:55:44 GMT
View Forum Message <> Reply to Message

On Mon, 2006-08-21 at 14:51 +0400, Kirill Korotaev wrote:
> Chandra Seetharaman wrote:
> > Kirill,
> >
> > IMO, a UBC with resource constraint(limit in this case) should behave no
> > different than a kernel with limited memory. i.e it should do
> > reclamation before it starts failing allocation requests. It could even
> > do it preemptively.
> first, please notice, that this thread is not about user memory.
> we can discuss it later when about to control user memory. And
> I still need to notice, that different models of user memory control
> can exist. With and without reclamation.
>
we can talk about it then :)

> > There is no guarantee support which is required for providing QoS.
> where? in UBC? in UBC _there_ are guarentees, even in regard to OOM killer.

I do not see it in the patches you have submitted. May be I overlooked.
Can you please point me the code where guarantee is handled.

>
> > Each controller modifying the infrastructure code doesn't look good. We
> > can have proper interfaces to add a new resource controller.
> controllers do not modify interfaces nor core. They just add
> themself to the list of resources and setup default limits.
> do you think it is worth creating infrastructure for these
> 2 one-line-changes?

Yes, IMO, it is cleaner.

Think of the documentation that explains how to write a controller for UBC.

With a proper interface it will read something like: One have to call register_controller(char *name) and on success it returns a unique id which is the id for the controller.

Vs

With changing lines in the core code: One have to edit the file filename.c and add a macro to this of macros with an incremented value for their controller and add the name of their controller to the array named controller_names[].

I think the first one is cleaner, what do you think ?

<snip>

--

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

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Chandra Seetharaman on Mon, 21 Aug 2006 21:45:21 GMT
View Forum Message <> Reply to Message

On Mon, 2006-08-21 at 17:24 +0400, Kirill Korotaev wrote:
> Chandra Seetharaman wrote:
> > Kirill,
> >
> > Here are some concerns I have (as of now) w.r.t using UBC for resource
> > management (in the context of resource groups).
> >
> > - guarantee support is missing. I do not see any code to provide the
> > minimum amount of resource a group can get. It is important for
> > providing QoS. (In a different email you did mention guarantee, i am
> > referring it here for completeness).
> I mentioned a couple of times that this is a limited core functionality
> in this patch set.
> guarantees are implementable as a separate UBC parameters.

I will wait for oomguarpages patches :)

>
> > - Creation of a UBC and assignment of task to a UBC always happen in
> > the context of the task that is affected. I can understand it works in
> > OpenVZ environment, but IMO has issues if one wants it to be used for
> > basic resource management
> >   - application needs to be changed to use this feature.
> >   - System administrator does not have the control to assign tasks to a
> >   UBC. Application does by itself.
> >   - Assignment of task to a UBC need to be transparent to the
> >   application.
> this is not 100% true.
> UBC itself doesn't prevent from changing context on the fly.
> But since this leads to part of resources to be charged to
> one UBC and another part to another UBC and so long and so

Let the controllers and the users worry about that part.

As I mentioned UBC might be perfect for container resource management,
but what I am talking for is resource management _without_ a container.

> forth, we believe that more clear and correct interface is
> something like fork()/exec()-required-application.
>
> So you can always execute new applications in desired UB and
> NO application modification are required.

For generic workload management/resource management _desired UB_ is not
necessarily decided at fork/exec time. It can happen anytime during the
life cycle of a task.

>
>
> > - UBC is deleted when the last task (in that UBC) exits. For resource
> > management purposes, UBC should be deleted only when the administrator
> > deletes it.
> 1. UBCs are freed when last _resource_ using it puts the last reference.
> not the task. And it is a BIG error IMHO to think that resource
> management should group tasks. No, it should group _objects_. Tasks
> are just the same objects like say sockets.

No argument there, that is how CKRM was early last year. But, I do not
see how is related to the point I am making above (" UBC should be
deleted only when the administrator deletes it").

> 2. this is easily changeable. You are the only who requested it so far.

It may be because I am the only one looking at it without the
"container" goggles on :).

> 3. kernel does so for many other objects like users and no one complains :)
>
> > - No ability to set resource specific configuration information.
> UBC model allows to _limit_ users. It is _core_.

I think you got me wrong here. What I want is the ability to
set/maintain a generic controller specific information.

For example, if the CPU controller wants to allow the user to set the
number of seconds over which the user wants the guarantee/limit to be
imposed.

> We want to do resource management step by step and send it patch by patch,
> while you are trying to solve everything at once.
>
> sys_open() for example doesn't allow to open sockets, does it?
> the same for UBC. They do what they are supposed to do.

I do not see how this relates !!

>
> > - No ability to maintain resource specific data in the controller.
> it's false. fields can be added to user_beancounter struct easily.
> and that's what our controllers do.

With the model of static array for resources (struct ubparm ub_parms
[UB_RESOURCES] in struct user_beancounter), it is not be possible to
attach _different_ "controller specific" information to each of the
entries.

I do not think it is good idea to add controller specific information of
different controllers to the user_beancounter. Think of all the fields
it will have when all the numproc controller needs is just the basic 3-4
fields.

>
> > - No ability to get the list of tasks belonging to a UBC.
> it is not true. it can be read from /proc or system calls interface,
> just like the way one finds all tasks belonging to one user :)
>
> BTW, what is so valueable in this feature?

Again, it may not be useful for container type usages (you can probably
get the list from somewhere else, but for resource management it is
useful for sysadmins).

> do you want to have interfaces to find kernel structures and even pages
> which belong to the container? tasks are just one type of objects...
>
> > - Doesn't inform the resource controllers when limits(shares) change.
> As was answered and noted by Alan Cox:
> 1. no one defined what type of action should be done when limits change

let the controller decide it.
> 2. it is extendable _when_ needed. Do you want to introduce hooks just
> to have them?
> 3. is it so BIG obstacle for UBC patch? These 3-lines hooks code which
> is not used?
>
> > - Doesn't inform the resource controllers when a task's UBC has changed.
> the same as above. we don't add functionality which is not used YET
> (and no one even knows HOW).
>
> > - Doesn't recalculate the resource usage when a task's UBC has changed.
> > i.e doesn't uncharge the old UBC and charge new UBC.
> You probably missed my explanation, that most
> resources (except for the simplest one - numproc) can't be recharged
> easily. And nothing in UBC code prevents such recharge to be added later
> if requested.

My point is that controllers should have this control. I am ok with
these being added later. Wondering if there is any design limitations
that would prevent the later additions (like the _controller specific
data above).

>
> > - For a system administrator name for identification of a UBC is
> > better than a number (uid).
> Have you any problems with pids, uids, gids and signals?

Again, in container land each UB is attached with a container hence no
issue.

In a non-container situation IMO it will be easier to manage/associate
"gold", "silver", "bronze", "plastic" groups than 0, 11, 83 and 113.

> It is a question of interface. I don't mind in changing UBC interface even
> to configfs if someone really wants it.
>
> Thanks,
> Kirill
>

> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Alan Cox on Mon, 21 Aug 2006 22:01:05 GMT
View Forum Message <> Reply to Message

Ar Llu, 2006-08-21 am 14:45 -0700, ysgrifennodd Chandra Seetharaman:
> As I mentioned UBC might be perfect for container resource management,
> but what I am talking for is resource management _without_ a container.

There isn't really a difference. UBC counts usage of things. It has to
know who to charge the thing to but its core concept of the luid isn't a
container, its more akin to the a departmental or project billing code.

> > 3. is it so BIG obstacle for UBC patch? These 3-lines hooks code which
> > is not used?

Add them later when they prove to be needed. If IBM send a feature that
needs it then add them in that feature. Everyone is happy it is possible
to add that hook when needed.

> In a non-container situation IMO it will be easier to manage/associate
> "gold", "silver", "bronze", "plastic" groups than 0, 11, 83 and 113.

User space issue. Doing that in kernel will lead to some limitations
later on and end up needing the user space anyway. Consider wanting to
keep the container name and properties in LDAP.

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Chandra Seetharaman on Mon, 21 Aug 2006 22:44:19 GMT
View Forum Message <> Reply to Message

On Mon, 2006-08-21 at 23:20 +0100, Alan Cox wrote:
> Ar Llu, 2006-08-21 am 14:45 -0700, ysgrifennodd Chandra Seetharaman:
> > As I mentioned UBC might be perfect for container resource management,
> > but what I am talking for is resource management _without_ a container.
>
> There isn't really a difference. UBC counts usage of things. It has to
> know who to charge the thing to but its core concept of the luid isn't a
> container, its more akin to the a departmental or project billing code.

I didn't say it is different. The way it is implemented now has some
restrictions for generic resource management purposes (like ability to
move task around), but they are not a problem for container type usage.

>
> > > 3. is it so BIG obstacle for UBC patch? These 3-lines hooks code which
> > >   is not used?
>
> Add them later when they prove to be needed. If IBM send a feature that
> needs it then add them in that feature. Everyone is happy it is possible
> to add that hook when needed.

As I mentioned in my reply, I am ok with adding it later.

>
> > In a non-container situation IMO it will be easier to manage/associate
> > "gold", "silver", "bronze", "plastic" groups than 0, 11, 83 and 113.
>
> User space issue. Doing that in kernel will lead to some limitations
> later on and end up needing the user space anyway. Consider wanting to
> keep the container name and properties in LDAP.
>
>
>
>
>
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

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

Subject: Re: [ckrm-tech] [PATCH 4/7] UBC: syscalls (user interface)
Posted by Rohit Seth on Tue, 22 Aug 2006 01:16:19 GMT
View Forum Message <> Reply to Message

On Mon, 2006-08-21 at 11:47 +0900, Magnus Damm wrote:
> On Fri, 2006-08-18 at 07:45 -0700, Dave Hansen wrote:
> > On Fri, 2006-08-18 at 12:08 +0400, Andrey Savochkin wrote:
> > >
> > > 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.
> >
> > Hold on here for just a sec...
> >
> > It is quite possible to do memory management aimed at one container
> > while that container's memory still participates in the main VM.
> >
> > There is overhead here, as the LRU scanning mechanisms get less
> > efficient, but I'd rather pay a penalty at LRU scanning time than divide
> > up the VM, or coarsely start failing allocations.
>
> This could of course be solved with one LRU per container, which is how
> the CKRM memory controller implemented things about a year ago.

Effectively Andrew's idea of faking up nodes is also giving per
container LRUs.

-rohit

Subject: Re: [RFC][PATCH 5/7] UBC: kernel memory accounting (core)
Posted by Rohit Seth on Tue, 22 Aug 2006 01:23:32 GMT
View Forum Message <> Reply to Message

On Mon, 2006-08-21 at 14:43 +0400, Kirill Korotaev wrote:
> >>1. reclaiming user resources is not that good idea as it looks to you.
> >>such solutions end up with lots of resources spent on reclaim.
> >>for user memory reclaims mean consumption of expensive disk I/O bandwidth
> >>which reduces overall system throughput and influences other users.
> >>
> >
> >
> > May be I'm overlooking something very obvious. Please tell me, what
> > happens when a user hits a page fault and the page allocator is easily
> > able to give a page from its pcp list. But container is over its limit
> > of physical memory. In your patch there is no attempt by container
> > support to see if some of the user pages are easily reclaimable. What

> > options a user will have to make sure some room is created.
> The patch set send doesn't control user memory!
> This topic is about kernel memory...
>

And that is why I asked the question in the very first mail (if this
support is going to come later).

-rohit

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Rohit Seth on Tue, 22 Aug 2006 01:45:28 GMT
View Forum Message <> Reply to Message

On Mon, 2006-08-21 at 14:45 -0700, Chandra Seetharaman wrote:
> On Mon, 2006-08-21 at 17:24 +0400, Kirill Korotaev wrote:
> > Chandra Seetharaman wrote:
> > > Kirill,
> > >
> > > Here are some concerns I have (as of now) w.r.t using UBC for resource
> > > management (in the context of resource groups).
> > >
> > > - guarantee support is missing. I do not see any code to provide the
> > >   minimum amount of resource a group can get. It is important for
> > >   providing QoS. (In a different email you did mention guarantee, i am
> > >   referring it here for completeness).
> > I mentioned a couple of times that this is a limited core functionality
> > in this patch set.
> > guarantees are implementable as a separate UBC parameters.
>
> I will wait for oomguarpages patches :)
>
> >
> > > - Creation of a UBC and assignment of task to a UBC always happen in
> > >   the context of the task that is affected. I can understand it works in
> > >   OpenVZ environment, but IMO has issues if one wants it to be used for
> > >   basic resource management
> > >   - application needs to be changed to use this feature.
> > >   - System administrator does not have the control to assign tasks to a
> > >   UBC. Application does by itself.
> > >   - Assignment of task to a UBC need to be transparent to the
> > >   application.

I agree with the above points. Just want to add that assignment of a
task to a container may not be transparent to the application. For
example it may hit some limits and some reclaim may happen...

> > this is not 100% true.
> > UBC itself doesn't prevent from changing context on the fly.
> > But since this leads to part of resources to be charged to
> > one UBC and another part to another UBC and so long and so
>
> Let the controllers and the users worry about that part.
>

I think as the tasks move around, it becomes very heavy to move all the
pages belonging to previous container to a new container.

> As I mentioned UBC might be perfect for container resource management,
> but what I am talking for is resource management _without_ a container.
>

Can you explain that part a bit more?

> >
> > > - No ability to maintain resource specific data in the controller.
> > it's false. fields can be added to user_beancounter struct easily.
> > and that's what our controllers do.
>
> With the model of static array for resources (struct ubparm ub_parms
> [UB_RESOURCES] in struct user_beancounter), it is not be possible to
> attach _different_ "controller specific" information to each of the
> entries.
>
> I do not think it is good idea to add controller specific information of
> _different_ controllers to the user_beancounter. Think of all the fields
> it will have when all the numproc controller needs is just the basic 3-4
> fields.
>

IMO it is okay to add the fields whenever necessary as Kirill
suggested. I think once the container subject gets baked a little more,
the controllers will also get tightly coupled.

> >
> > > - No ability to get the list of tasks belonging to a UBC.
> > it is not true. it can be read from /proc or system calls interface,
> > just like the way one finds all tasks belonging to one user :)
> >
> > BTW, what is so valueable in this feature?
>
> Again, it may not be useful for container type usages (you can probably
> get the list from somewhere else, but for resource management it is
> useful for sysadmins).

>

I'm also debating about whether printing task information is really any
useful. If a sysadm wants to get information about any particular task
then that can come from /proc/<pid>/container Though container list
will be one place where one can easily get the list of all the contained
tasks (and other resources like files).

> >
> > > - For a system administrator name for identification of a UBC is
> > >   better than a number (uid).
> > Have you any problems with pids, uids, gids and signals?
>
> Again, in container land each UB is attached with a container hence no
> issue.
>
> In a non-container situation IMO it will be easier to manage/associate
> "gold", "silver", "bronze", "plastic" groups than 0, 11, 83 and 113.
>
>
> > It is a question of interface. I don't mind in changing UBC interface even
> > to configfs if someone really wants it.
> >

Yes please. Thanks.
-rohit

Subject: Re: [ckrm-tech] [PATCH 4/7] UBC: syscalls (user interface)
Posted by Magnus Damm on Tue, 22 Aug 2006 03:58:06 GMT
View Forum Message <> Reply to Message

On Mon, 2006-08-21 at 18:16 -0700, Rohit Seth wrote:
> On Mon, 2006-08-21 at 11:47 +0900, Magnus Damm wrote:
> > On Fri, 2006-08-18 at 07:45 -0700, Dave Hansen wrote:
> > > On Fri, 2006-08-18 at 12:08 +0400, Andrey Savochkin wrote:
> > > >
> > > > 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.
> > >
> > > Hold on here for just a sec...
> > >
> > > It is quite possible to do memory management aimed at one container
> > > while that container's memory still participates in the main VM.
> > >

> > > There is overhead here, as the LRU scanning mechanisms get less
> > > efficient, but I'd rather pay a penalty at LRU scanning time than divide
> > > up the VM, or coarsely start failing allocations.
> >
> > This could of course be solved with one LRU per container, which is how
> > the CKRM memory controller implemented things about a year ago.
>
> Effectively Andrew's idea of faking up nodes is also giving per
> container LRUs.

Yes, but the NUMA emulation approach is using fixed size containers
where the size is selectable at the kernel command line, while the CKRM
(and pzone) approach provides a more dynamic (and complex) solution.

/ magnus

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Alan Cox on Tue, 22 Aug 2006 09:42:03 GMT
View Forum Message <> Reply to Message

Ar Llu, 2006-08-21 am 18:45 -0700, ysgrifennodd Rohit Seth:
> I think as the tasks move around, it becomes very heavy to move all the
> pages belonging to previous container to a new container.

Its not a meaningful thing to do. Remember an object may be passed
around or shared. The simple "creator pays" model avoids all the heavy
overheads while maintaining the constraints.

Its only user space pages that some of this (AS and RSS) become
interesting as "movable" objects

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Arjan van de Ven on Tue, 22 Aug 2006 09:57:48 GMT
View Forum Message <> Reply to Message

On Tue, 2006-08-22 at 11:02 +0100, Alan Cox wrote:
> Ar Llu, 2006-08-21 am 18:45 -0700, ysgrifennodd Rohit Seth:
> > I think as the tasks move around, it becomes very heavy to move all the
> > pages belonging to previous container to a new container.
>
> Its not a meaningful thing to do. Remember an object may be passed
> around or shared. The simple "creator pays" model avoids all the heavy
> overheads while maintaining the constraints.

Hi,

there is one issue with the "creator pays" model: if the creator can
decide to die/go away/respawn then you can create orphan resources. This
is a leak at least, but if a malicious user can control the
death/respawn cycle it can even be abused to bypass the controls in the
first place. Keeping the owner alive until all shared users are gone is
not always a good idea either; if a container significantly malfunctions
(or requires a restart due to, say, a very urgent glibc security
update), keeping it around anyway is not a valid option for the admin.
(And it forms another opportunity for a malicious user, keep a
(vulnerable) container alive by hanging on to a shared resource
deliberately)

A general "unshare me out" function that finds a new to-blame owner
might work, just the decision whom to blame is not an easy one in that
scenario.

Greetings,
  Arjan van de Ven

--
if you want to mail me at work (you don't), use arjan (at) linux.intel.com

---

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Alan Cox on Tue, 22 Aug 2006 10:54:20 GMT
View Forum Message <> Reply to Message

Ar Maw, 2006-08-22 am 11:57 +0200, ysgrifennodd Arjan van de Ven:
> there is one issue with the "creator pays" model: if the creator can
> decide to die/go away/respawn then you can create orphan resources. This

You cannot create orphan resources with UBC. All resources have an
owner. You might be able to construct a hypothetical scenario where I
commit all my resources to other people but I cannot create orphan
resources or leak them.

Even if I am the only user of a given UBC my counter will survive until
the last object is freed, not until I log out. If I log back in my
resource accounting is still there and nothing has escaped.

Alan

Subject: Re: [ckrm-tech] [PATCH 4/7] UBC: syscalls (user interface)
Posted by Chandra Seetharaman on Tue, 22 Aug 2006 18:34:18 GMT

On Tue, 2006-08-22 at 12:58 +0900, Magnus Damm wrote:
> On Mon, 2006-08-21 at 18:16 -0700, Rohit Seth wrote:
> > On Mon, 2006-08-21 at 11:47 +0900, Magnus Damm wrote:
> > > On Fri, 2006-08-18 at 07:45 -0700, Dave Hansen wrote:
> > > > On Fri, 2006-08-18 at 12:08 +0400, Andrey Savochkin wrote:
> > > > >
> > > > > 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.
> > > > >
> > > > Hold on here for just a sec...
> > > >
> > > > It is quite possible to do memory management aimed at one container
> > > > while that container's memory still participates in the main VM.
> > > >
> > > > There is overhead here, as the LRU scanning mechanisms get less
> > > > efficient, but I'd rather pay a penalty at LRU scanning time than divide
> > > > up the VM, or coarsely start failing allocations.
> > >
> > > This could of course be solved with one LRU per container, which is how
> > > the CKRM memory controller implemented things about a year ago.
> >
> > Effectively Andrew's idea of faking up nodes is also giving per
> > container LRUs.
>
> Yes, but the NUMA emulation approach is using fixed size containers
> where the size is selectable at the kernel command line, while the CKRM
> (and pzone) approach provides a more dynamic (and complex) solution.

NUMA emulation does not allow guarantee, only limits. It also doesn't
allow over commit (ove commit issue is present in pzone based approach
also).

>
> / magnus
>
>
>
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Chandra Seetharaman on Tue, 22 Aug 2006 18:55:02 GMT
View Forum Message <> Reply to Message

On Mon, 2006-08-21 at 18:45 -0700, Rohit Seth wrote:
> On Mon, 2006-08-21 at 14:45 -0700, Chandra Seetharaman wrote:
> > On Mon, 2006-08-21 at 17:24 +0400, Kirill Korotaev wrote:
> > > Chandra Seetharaman wrote:
> > > > Kirill,
> > > >
> > > > Here are some concerns I have (as of now) w.r.t using UBC for resource
> > > > management (in the context of resource groups).
> > > >
> > > > - guarantee support is missing. I do not see any code to provide the
> > > > minimum amount of resource a group can get. It is important for
> > > > providing QoS. (In a different email you did mention guarantee, i am
> > > > referring it here for completeness).
> > > I mentioned a couple of times that this is a limited core functionality
> > > in this patch set.
> > > guarantees are implementable as a separate UBC parameters.
> >
> > I will wait for oomguarpages patches :)
> >
> > >
> > > > - Creation of a UBC and assignment of task to a UBC always happen in
> > > > the context of the task that is affected. I can understand it works in
> > > > OpenVZ environment, but IMO has issues if one wants it to be used for
> > > > basic resource management
> > > > - application needs to be changed to use this feature.
> > > > - System administrator does not have the control to assign tasks to a
> > > > UBC. Application does by itself.
> > > > - Assignment of task to a UBC need to be transparent to the
> > > > application.
>
> I agree with the above points. Just want to add that assignment of a
> task to a container may not be transparent to the application. For
> example it may hit some limits and some reclaim may happen...

By transparent I meant that the task _need_ not have to know that there
is a resource manager sitting and managing its resources. Task will
still see the effects of resource crunch etc., (but it will handle the

situation the same way as it would handle today).

So, it is transparent. A task don't have to know that the reclamation is happening due to its affiliation to a resource group. Task will be handling it as if there is a pressure for that particular resource.

>
> > > this is not 100% true.
> > > UBC itself doesn't prevent from changing context on the fly.
> > > But since this leads to part of resources to be charged to
> > > one UBC and another part to another UBC and so long and so
> >
> > Let the controllers and the users worry about that part.
> >
>
> I think as the tasks move around, it becomes very heavy to move all the
> pages belonging to previous container to a new container.

Not for all resources, CPU could handle it very nicely, whereas memory would be heavy. My point is that the infrastructure should be open, and controller is the one that decides whether it wants to handle it or not.

>
> > As I mentioned UBC might be perfect for container resource management,
> > but what I am talking for is resource management _without_ a container.
> >
>
> Can you explain that part a bit more?

Basically I was saying that even though resource management in container and non-container have mostly same requirements, there are few requirements that are critical in non-container scenario which are non-issue in container scenario (for example, moving tasks from one resource group to another).

In effect, Design of the infrastructure should not limit non-container usages.

IMO, non-container requirements are a superset of container requirements (resource management purposes only :).

>
> > >
> > > > - No ability to maintain resource specific data in the controller.
> > > it's false. fields can be added to user_beancounter struct easily.
> > > and that's what our controllers do.
> >
> > With the model of static array for resources (struct ubparm ub_parms

> > [UB_RESOURCES] in struct user_beancounter), it is not be possible to
> > attach _different_ "controller specific" information to each of the
> > entries.
> >
> > I do not think it is good idea to add controller specific information of
> > _different_ controllers to the user_beancounter. Think of all the fields
> > it will have when all the numproc controller needs is just the basic 3-4
> > fields.
> >
>
> IMO it is okay to add the fields whenever necessary as Kirill
> suggested. I think once the container subject gets baked a little more,
> the controllers will also get tightly coupled.

I think my point is not understood. I do not think it is right to add _controller specific_ fields to the generic data structure (struct user_beancounter), as we will end up with a generic data structure which will have so many fields that are not used in so many controllers.

>
> > >
> > > > - No ability to get the list of tasks belonging to a UBC.
> > > it is not true. it can be read from /proc or system calls interface,
> > > just like the way one finds all tasks belonging to one user :)
> > >
> > > BTW, what is so valueable in this feature?
> >
> > Again, it may not be useful for container type usages (you can probably
> > get the list from somewhere else, but for resource management it is
> > useful for sysadmins).
> >
>
> I'm also debating about whether printing task information is really any
> useful. If a sysadm wants to get information about any particular task
> then that can come from /proc/<pid>/container Though container list
> will be one place where one can easily get the list of all the contained
> tasks (and other resources like files).

In non-container environment, there is _no_ /proc/pid/container, as there is no concept of container :). This will be useful for non-container scenario.

<snip>
--

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

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

Subject: Re: [ckrm-tech] [PATCH 4/7] UBC: syscalls (user interface)
Posted by Rohit Seth on Thu, 24 Aug 2006 01:20:35 GMT
View Forum Message <> Reply to Message

On Tue, 2006-08-22 at 12:58 +0900, Magnus Damm wrote:
> On Mon, 2006-08-21 at 18:16 -0700, Rohit Seth wrote:
> > On Mon, 2006-08-21 at 11:47 +0900, Magnus Damm wrote:
> > > On Fri, 2006-08-18 at 07:45 -0700, Dave Hansen wrote:
> > > > On Fri, 2006-08-18 at 12:08 +0400, Andrey Savochkin wrote:
> > > >
> > > > 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.
> > > >
> > > > Hold on here for just a sec...
> > > >
> > > > It is quite possible to do memory management aimed at one container
> > > > while that container's memory still participates in the main VM.
> > > >
> > > > There is overhead here, as the LRU scanning mechanisms get less
> > > > efficient, but I'd rather pay a penalty at LRU scanning time than divide
> > > > up the VM, or coarsely start failing allocations.
> > >
> > > This could of course be solved with one LRU per container, which is how
> > > the CKRM memory controller implemented things about a year ago.
> >
> > Effectively Andrew's idea of faking up nodes is also giving per
> > container LRUs.
>
> Yes, but the NUMA emulation approach is using fixed size containers
> where the size is selectable at the kernel command line,
[Apologies for late reply..]

Yup, if we run with fake NUMA support for providing container
functionality then dynamic resizing will be important (and that is why I
made the initial comment of possibly using memory hot-plug)

> while the CKRM
> (and pzone) approach provides a more dynamic (and complex) solution.

...this complexity is not always a positive thing ;-) (I do like core
of CKRM stuff FWIW).

-rohit

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Rohit Seth on Thu, 24 Aug 2006 01:31:26 GMT
View Forum Message <> Reply to Message

On Tue, 2006-08-22 at 11:02 +0100, Alan Cox wrote:
> Ar Llu, 2006-08-21 am 18:45 -0700, ysgrifennodd Rohit Seth:
> > I think as the tasks move around, it becomes very heavy to move all the
> > pages belonging to previous container to a new container.
>
> Its not a meaningful thing to do. Remember an object may be passed
> around or shared. The simple "creator pays" model avoids all the heavy
> overheads while maintaining the constraints.
>

I agree, creator pays model will be good for anonymous pages. (And this
is where page based container will help).

> Its only user space pages that some of this (AS and RSS) become
> interesting as "movable" objects
>

I think something like for AS, yes. But for anonymous pages, might want
to leave them back.

-rohit

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Rohit Seth on Thu, 24 Aug 2006 01:44:41 GMT
View Forum Message <> Reply to Message

On Tue, 2006-08-22 at 11:55 -0700, Chandra Seetharaman wrote:
> On Mon, 2006-08-21 at 18:45 -0700, Rohit Seth wrote:
> >
> > > > this is not 100% true.
> > > > UBC itself doesn't prevent from changing context on the fly.
> > > > But since this leads to part of resources to be charged to
> > > > one UBC and another part to another UBC and so long and so
> > >
> > > Let the controllers and the users worry about that part.
> > >
> >
> > I think as the tasks move around, it becomes very heavy to move all the
> > pages belonging to previous container to a new container.

>
> Not for all resources, CPU could handle it very nicely, whereas memory
> would be heavy. My point is that the infrastructure should be open, and
> controller is the one that decides whether it wants to handle it or not.

With open you are implying being able to use different ones. It would
be nice to get one in and make sure it is stable and optimized...

>
> >
> > > As I mentioned UBC might be perfect for container resource management,
> > > but what I am talking for is resource management _without_ a container.
> > >
> >
> > Can you explain that part a bit more?
>
> Basically I was saying that even though resource management in container
> and non-container have mostly same requirements, there are few
> requirements that are critical in non-container scenario which are non-
> issue in container scenario (for example, moving tasks from one resource
> group to another).
>
> In effect, Design of the infrastructure should not limit non-container
> usages.
>
> IMO, non-container requirements are a superset of container requirements
> (resource management purposes only :).
>

hmm, non-container world (and its resource management part) already
exist. And sure those requirements are superset of this discussion.
And hopefully container support will not break/modify that much.

> >
> > > >
> > > > > - No ability to maintain resource specific data in the controller.
> > > > it's false. fields can be added to user_beancounter struct easily.
> > > > and that's what our controllers do.
> > >
> > > With the model of static array for resources (struct ubparm ub_parms
> > > [UB_RESOURCES] in struct user_beancounter), it is not be possible to
> > > attach _different_ "controller specific" information to each of the
> > > entries.
> > >
> > > I do not think it is good idea to add controller specific information of
> > > _different_ controllers to the user_beancounter. Think of all the fields
> > > it will have when all the numproc controller needs is just the basic 3-4
> > > fields.

> > >
> >
> > IMO it is okay to add the fields whenever necessary as Kirill
> > suggested. I think once the container subject gets baked a little more,
> > the controllers will also get tightly coupled.
>
> I think my point is not understood. I do not think it is right to add
> _controller specific_ fields to the generic data structure (struct
> user_beancounter), as we will end up with a generic data structure which
> will have so many fields that are not used in so many controllers.
>

A single centralized structure that has fields that are mostly used by
every one should be okay I think.

> >
> > > >
> > > > > - No ability to get the list of tasks belonging to a UBC.
> > > > it is not true. it can be read from /proc or system calls interface,
> > > > just like the way one finds all tasks belonging to one user :)
> > > >
> > > > BTW, what is so valueable in this feature?
> > >
> > > Again, it may not be useful for container type usages (you can probably
> > > get the list from somewhere else, but for resource management it is
> > > useful for sysadmins).
> > >
> >
> > I'm also debating about whether printing task information is really any
> > useful. If a sysadm wants to get information about any particular task
> > then that can come from /proc/<pid>/container Though container list
> > will be one place where one can easily get the list of all the contained
> > tasks (and other resources like files).
>
> In non-container environment, there is _no_ /proc/pid/container, as
> there is no concept of container :). This will be useful for non-
> container scenario.
>

I'm sure when container support gets in then for the above scenario it
will read -1 ...

-rohit

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Chandra Seetharaman on Thu, 24 Aug 2006 02:04:32 GMT

On Wed, 2006-08-23 at 18:44 -0700, Rohit Seth wrote:
> On Tue, 2006-08-22 at 11:55 -0700, Chandra Seetharaman wrote:
> > On Mon, 2006-08-21 at 18:45 -0700, Rohit Seth wrote:
> > >
> > > > > > > this is not 100% true.
> > > > > > > UBC itself doesn't prevent from changing context on the fly.
> > > > > > > But since this leads to part of resources to be charged to
> > > > > > > one UBC and another part to another UBC and so long and so
> > > > >
> > > > > Let the controllers and the users worry about that part.
> > > > >
> > > >
> > > > I think as the tasks move around, it becomes very heavy to move all the
> > > > pages belonging to previous container to a new container.
> > >
> > > Not for all resources, CPU could handle it very nicely, whereas memory
> > > would be heavy. My point is that the infrastructure should be open, and
> > > controller is the one that decides whether it wants to handle it or not.
> >
> > With open you are implying being able to use different ones. It would
> > be nice to get one in and make sure it is stable and optimized...

No, what I mean is that the infrastructure should allow the task moving
from one group to another, it should also notify the controller about
that movement and let the controller decide if it wants to take any
action. (instead of not having the capability stating that it is not
useful for all type of controllers).

> >
> > >
> > > > > > > As I mentioned UBC might be perfect for container resource management,
> > > > > > > but what I am talking for is resource management _without_ a container.
> > > > >
> > > > > Can you explain that part a bit more?
> > > >
> > > > Basically I was saying that even though resource management in container
> > > > and non-container have mostly same requirements, there are few
> > > > requirements that are critical in non-container scenario which are non-
> > > > issue in container scenario (for example, moving tasks from one resource
> > > > group to another).
> > > >
> > > > In effect, Design of the infrastructure should not limit non-container
> > > > usages.
> > > >
> > > > IMO, non-container requirements are a superset of container requirements

> > (resource management purposes only :).
> >
>
> hmm, non-container world (and its resource management part) already
> exist. And sure those requirements are superset of this discussion.

What do you mean by "resource management part for non-container world
already exist ?

It does not. CKRM/Resource Groups is trying to do that, but is not in
Linus's tree.

> And hopefully container support will not break/modify that much.
>
> > >
> > > > >
> > > > > > - No ability to maintain resource specific data in the controller.
> > > > > > it's false. fields can be added to user_beancounter struct easily.
> > > > > > and that's what our controllers do.
> > > > >
> > > > > With the model of static array for resources (struct ubparm ub_parms
> > > > > [UB_RESOURCES] in struct user_beancounter), it is not be possible to
> > > > > attach _different_ "controller specific" information to each of the
> > > > > entries.
> > > > >
> > > > > I do not think it is good idea to add controller specific information of
> > > > > _different_ controllers to the user_beancounter. Think of all the fields
> > > > > it will have when all the numproc controller needs is just the basic 3-4
> > > > > fields.
> > > > >
> > > >
> > > > IMO it is okay to add the fields whenever necessary as Kirill
> > > > suggested. I think once the container subject gets baked a little more,
> > > > the controllers will also get tightly coupled.
> > >
> > > I think my point is not understood. I do not think it is right to add
> > > _controller specific_ fields to the generic data structure (struct
> > > user_beancounter), as we will end up with a generic data structure which
> > > will have so many fields that are not used in so many controllers.
> > >
> >
> > A single centralized structure that has fields that are mostly used by
> > every one should be okay I think.

You mean to say definition like

```
struct user_beancounter {
 fields;/* fields that exists now */
```

```
 int kmemsize_ctlr_info1;
 char *kmemsize_ctlr_info2;

 char *oomguar_ctlr_info1;
 char *oomguar_ctlr_info2;

 /* and so on */
}
```

is the right thing to do ? even though oomguar controller doesn't care
about kmemsize_ctlr_info* etc.,

> > > > > > > > > > > - No ability to get the list of tasks belonging to a UBC.
> > > > > > it is not true. it can be read from /proc or system calls interface,
> > > > > > just like the way one finds all tasks belonging to one user :)
> > > > > > BTW, what is so valueable in this feature?
> > > > > Again, it may not be useful for container type usages (you can probably
> > > > > get the list from somewhere else, but for resource management it is
> > > > > useful for sysadmins).
> > > > > > > I'm also debating about whether printing task information is really any
> > > > > useful. If a sysadm wants to get information about any particular task
> > > > > then that can come from /proc/<pid>/container Though container list
> > > > > will be one place where one can easily get the list of all the contained
> > > > > tasks (and other resources like files).
> > > > > In non-container environment, there is _no_ /proc/pid/container, as
> > there is no concept of container :). This will be useful for non-
> > container scenario.
> > > > I'm sure when container support gets in then for the above scenario it
> will read -1 ...

So, how can one get the list of tasks belonging to a resource group in
that case ?

> -rohit
> > > ---------- ----------

> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Alan Cox on Thu, 24 Aug 2006 10:49:06 GMT
View Forum Message <> Reply to Message

Ar Mer, 2006-08-23 am 19:04 -0700, ysgrifennodd Chandra Seetharaman:
> > A single centralized structure that has fields that are mostly used by
> > every one should be okay I think.
>
> You mean to say definition like
>
> struct user_beancounter {
> fields;/* fields that exists now */
>
> int kmemsize_ctlr_info1;
> char *kmemsize_ctlr_info2;
>
> char *oomguar_ctlr_info1;
> char *oomguar_ctlr_info2;
>
> /* and so on */
> }
>
> is the right thing to do ? even though oomguar controller doesn't care
> about kmemsize_ctlr_info* etc.,

All you need is

struct wombat_controller
{
 struct user_beancounter counter;
 void (*wombat_pest_control)(struct wombat *w);
 atomic_t wombat_population;

```
 int (*wombat_destructor)(struct wombat *w);
};
```

and just embed the counter in whatever you are controlling. The point of the beancounters themselves is to be *SIMPLE*. It's unfortunate that some folk seem obsessed with extending them for a million theoretical projects rather than getting them in and working and then extending them for real projects. Please lets not have another EVMS.

Alan

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Rohit Seth on Thu, 24 Aug 2006 17:27:41 GMT
View Forum Message <> Reply to Message

On Wed, 2006-08-23 at 19:04 -0700, Chandra Seetharaman wrote:
> On Wed, 2006-08-23 at 18:44 -0700, Rohit Seth wrote:

> No, what I mean is that the infrastructure should allow the task moving
> from one group to another, it should also notify the controller about
> that movement and let the controller decide if it wants to take any
> action. (instead of not having the capability stating that it is not
> useful for all type of controllers).
>
> >
Okay.

> > >
> > > >
> > > > > As I mentioned UBC might be perfect for container resource management,
> > > > > but what I am talking for is resource management _without_ a container.
> > > > >
> > > >
> > > > Can you explain that part a bit more?
> > >
> > > Basically I was saying that even though resource management in container
> > > and non-container have mostly same requirements, there are few
> > > requirements that are critical in non-container scenario which are non-
> > > issue in container scenario (for example, moving tasks from one resource
> > > group to another).
> > >
> > > In effect, Design of the infrastructure should not limit non-container
> > > usages.
> > >
> > > IMO, non-container requirements are a superset of container requirements
> > > (resource management purposes only :).
> > >

> >
> > hmm, non-container world (and its resource management part) already
> > exist. And sure those requirements are superset of this discussion.
>
> What do you mean by "resource management part for non-container world
> already exist ?
>
> It does not. CKRM/Resource Groups is trying to do that, but is not in
> Linus's tree.
>

Please, non-container is the environment that exist today in Linux.
Actually cpuset does provide some part of it. But beyond that no.

But then we are all using different terminology like beancounters,
containers, resource groups and now non-containers...

> > And hopefully container support will not break/modify that much.
> >
> > > > >
> > > > > > >
> > > > > > > > - No ability to maintain resource specific data in the controller.
> > > > > > > it's false. fields can be added to user_beancounter struct easily.
> > > > > > > and that's what our controllers do.
> > > > > > >
> > > > > > > With the model of static array for resources (struct ubparm ub_parms
> > > > > > > [UB_RESOURCES] in struct user_beancounter), it is not be possible to
> > > > > > > attach _different_ "controller specific" information to each of the
> > > > > > > entries.
> > > > > > >
> > > > > > > I do not think it is good idea to add controller specific information of
> > > > > > > _different_ controllers to the user_beancounter. Think of all the fields
> > > > > > > it will have when all the numproc controller needs is just the basic 3-4
> > > > > > > fields.
> > > > > > >
> > > > > >
> > > > > > IMO it is okay to add the fields whenever necessary as Kirill
> > > > > > suggested. I think once the container subject gets baked a little more,
> > > > > > the controllers will also get tightly coupled.
> > > > >
> > > > > I think my point is not understood. I do not think it is right to add
> > > > > _controller specific_ fields to the generic data structure (struct
> > > > > user_beancounter), as we will end up with a generic data structure which
> > > > > will have so many fields that are not used in so many controllers.
> > > >
> > >
> > > A single centralized structure that has fields that are mostly used by

> > every one should be okay I think.
>
> You mean to say definition like
>
> struct user_beancounter {
> fields;/* fields that exists now */
>
> int kmemsize_ctlr_info1;
> char *kmemsize_ctlr_info2;
>
> char *oomguar_ctlr_info1;
> char *oomguar_ctlr_info2;
>
> /* and so on */
> }
>
> is the right thing to do ? even though oomguar controller doesn't care
> about kmemsize_ctlr_info* etc.,
>

No. I think it is appropriate to add all the accounting related fields
and object fields in the core container definition. Controllers only
make decisions based on the information contained in container. And it
should maintain its own data structures if needed(like what Alan said in
one of the later mails).

> >
> > > >
> > > > > >
> > > > > > > - No ability to get the list of tasks belonging to a UBC.
> > > > > > it is not true. it can be read from /proc or system calls interface,
> > > > > > just like the way one finds all tasks belonging to one user :)
> > > > > >
> > > > > > BTW, what is so valueable in this feature?
> > > > >
> > > > > Again, it may not be useful for container type usages (you can probably
> > > > > get the list from somewhere else, but for resource management it is
> > > > > useful for sysadmins).
> > > > >
> > > >
> > > > I'm also debating about whether printing task information is really any
> > > > useful. If a sysadm wants to get information about any particular task
> > > > then that can come from /proc/<pid>/container Though container list
> > > > will be one place where one can easily get the list of all the contained
> > > > tasks (and other resources like files).
> > >
> > > In non-container environment, there is _no_ /proc/pid/container, as
> > > there is no concept of container :). This will be useful for non-

> > > container scenario.
> > >
> >
> > I'm sure when container support gets in then for the above scenario it
> > will read -1 ...
>
> So, how can one get the list of tasks belonging to a resource group in
> that case ?
> >

...and that brings to the starting question...why do you need it?
Commands like ps and top will show appropriate container number for each task.

-rohit

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Chandra Seetharaman on Thu, 24 Aug 2006 23:48:28 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-24 at 12:10 +0100, Alan Cox wrote:
> Ar Mer, 2006-08-23 am 19:04 -0700, ysgrifennodd Chandra Seetharaman:
> > > A single centralized structure that has fields that are mostly used by
> > > every one should be okay I think.
> >
> > You mean to say definition like
> >
> > struct user_beancounter {
> > fields;/* fields that exists now */
> >
> > int kmemsize_ctlr_info1;
> > char *kmemsize_ctlr_info2;
> >
> > char *oomguar_ctlr_info1;
> > char *oomguar_ctlr_info2;
> >
> > /* and so on */
> > }
> >
> > is the right thing to do ? even though oomguar controller doesn't care
> > about kmemsize_ctlr_info* etc.,
>
>
> All you need is
>
> struct wombat_controller
> {

> struct user_beancounter counter;
> void (*wombat_pest_control)(struct wombat *w);
> atomic_t wombat_population;
> int (*wombat_destructor)(struct wombat *w);
> };

This may not solve the problem, as
 - we won't be able get the controller data structure given the
   beancounter data structure.
 - we need to keep the data in sync (since there are multiple copies).
 - we will be copying the whole beancounter data structure needlessly
   (the controller might care only about _its_ parameters).

I agree with you that this can be added later when needed. The problem I see is that this might need some change in the core data structure which might face more resistance (than it does now :) once it is in mainline.

>
> and just embed the counter in whatever you are controlling. The point of
> the beancounters themselves is to be *SIMPLE*. It's unfortunate that
> some folk seem obsessed with extending them for a million theoretical
> projects rather than getting them in and working and then extending them
> for real projects. Please lets not have another EVMS.
>
> Alan
>
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Chandra Seetharaman on Thu, 24 Aug 2006 23:52:52 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-24 at 10:27 -0700, Rohit Seth wrote:

<snip>

> > What do you mean by "resource management part for non-container world
> > already exist ?
> >
> > It does not. CKRM/Resource Groups is trying to do that, but is not in
> > Linus's tree.
> >

>
> Please, non-container is the environment that exist today in Linux.
> Actually cpuset does provide some part of it. But beyond that no.

cpuset provides resource _isolation_, not necessarily resource
management.

>
> But then we are all using different terminology like beancounters,
> containers, resource groups and now non-containers...
>

<snip>

> > > I'm sure when container support gets in then for the above scenario it
> > > will read -1 ...
> >
> > So, how can one get the list of tasks belonging to a resource group in
> > that case ?
> > >
>
> ...and that brings to the starting question...why do you need it?

Like I said earlier, there is _no_ other way to get the list of tasks
belonging to a resource group.

> Commands like ps and top will show appropriate container number for each
> task.

There is _no_ container number in the non-container environment (or it
will be same for _all_ tasks).

>
> -rohit
>
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Kyle Moffett on Thu, 24 Aug 2006 23:55:11 GMT
View Forum Message <> Reply to Message

On Aug 24, 2006, at 19:48:28, Chandra Seetharaman wrote:

> On Thu, 2006-08-24 at 12:10 +0100, Alan Cox wrote:
>> All you need is
>>
>> struct wombat_controller
>> {
>> struct user_beancounter counter;
>> void (*wombat_pest_control)(struct wombat *w);
>> atomic_t wombat_population;
>> int (*wombat_destructor)(struct wombat *w);
>> };
>
> This may not solve the problem, as
> - we won't be able get the controller data structure given the
> beancounter data structure.

Of course you can! This is what we do for linked lists too. Here's
an example of how to get a pointer to your wombat_controller given
the user_beancounter pointer:
struct wombat_controller *wombat = containerof
(ptr_to_user_beancounter, struct wombat_controller, counter);

The containerof(PTR, TYPE, MEMBER) returns a pointer to the parent
object of type "TYPE" whose member "MEMBER" has address "PTR".

Cheers,
Kyle Moffett

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by dev on Fri, 25 Aug 2006 11:10:15 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:
> On Thu, 2006-08-24 at 10:27 -0700, Rohit Seth wrote:
>
> <snip>
>
>>>What do you mean by "resource management part for non-container world
>>>already exist ?
>>>
>>>It does not. CKRM/Resource Groups is trying to do that, but is not in
>>>Linus's tree.
>>>
>>
>>Please, non-container is the environment that exist today in Linux.
>>Actually cpuset does provide some part of it. But beyond that no.
>
>

> cpuset provides resource _isolation_, not necessarily resource
> management.
>
>
>>But then we are all using different terminology like beancounters,
>>containers, resource groups and now non-containers...
>>
>
>
> <snip>
>
>>>>I'm sure when container support gets in then for the above scenario it
>>>>will read -1 ...
>>>
>>>So, how can one get the list of tasks belonging to a resource group in
>>>that case ?
>>>
>>...and that brings to the starting question...why do you need it?
>
>
> Like I said earlier, there is _no_ other way to get the list of tasks
> belonging to a resource group.
>
>
>>Commands like ps and top will show appropriate container number for each
>>task.
>
>
> There is _no_ container number in the non-container environment (or it
> will be same for _all_ tasks).

Chandra, virtual container number is essentially the same as user id
in non-container environment. UBC were desgined for _users_ first.
Containers were just the first environment which started to use it widely.

And I really disagree when you say that non-container usecase is
a superset of container usecase. I believe it is vice versa, since
in container usecase you have a _full_ environment with root user and need
more resources to be taken into account.

Thanks,
Kirill

Subject: Re: [RFC][PATCH 1/7] UBC: kconfig
Posted by Pavel Machek on Fri, 25 Aug 2006 15:12:58 GMT
View Forum Message <> Reply to Message

Hi!

> --- ./kernel/ub/Kconfig.ubkm 2006-07-28
> 13:07:38.000000000 +0400
> +++ ./kernel/ub/Kconfig 2006-07-28
> 13:09:51.000000000 +0400
> @@ -0,0 +1,25 @@
> +#
> +# User resources part (UBC)
> +#
> +# Copyright (C) 2006 OpenVZ. SWsoft Inc

If you add copyright, add GPL, too.

> +config USER_RESOURCE
> + bool "Enable user resource accounting"
> + default y

New features should be disabled by default.

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

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Chandra Seetharaman on Fri, 25 Aug 2006 18:21:48 GMT
View Forum Message <> Reply to Message

On Thu, 2006-08-24 at 19:55 -0400, Kyle Moffett wrote:
> On Aug 24, 2006, at 19:48:28, Chandra Seetharaman wrote:
> > On Thu, 2006-08-24 at 12:10 +0100, Alan Cox wrote:
> >> All you need is
> >>
> >> struct wombat_controller
> >> {
> >> struct user_beancounter counter;
> >> void (*wombat_pest_control)(struct wombat *w);
> >> atomic_t wombat_population;
> >> int (*wombat_destructor)(struct wombat *w);
> >> };
> >
> > This may not solve the problem, as
> > - we won't be able get the controller data structure given the
> > beancounter data structure.
>
> Of course you can! This is what we do for linked lists too. Here's
> an example of how to get a pointer to your wombat_controller given

> the user_beancounter pointer:
> struct wombat_controller *wombat = containerof
> (ptr_to_user_beancounter, struct wombat_controller, counter);
>
> The containerof(PTR, TYPE, MEMBER) returns a pointer to the parent
> object of type "TYPE" whose member "MEMBER" has address "PTR".

Yes, it would work nicely.

But, the problem is that the struct user_beancounter (part of
wombat_controller above) is a _copy_ of the original, not the original
itself. We cannot keep the original (in _each_ controller), as there may
be more than one controller in the system and user_beancounter structure
is created/owned/destroyed by the beancounter infrastructure and not the
controller.

> Cheers,
> Kyle Moffett
>
>
>
>
> ---
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Chandra Seetharaman on Fri, 25 Aug 2006 18:47:24 GMT
View Forum Message <> Reply to Message

On Fri, 2006-08-25 at 15:12 +0400, Kirill Korotaev wrote:
<snip>
> >
> >
> > Like I said earlier, there is _no_ other way to get the list of tasks
> > belonging to a resource group.

> >
> >
> >>Commands like ps and top will show appropriate container number for each
> >>task.
> >
> >
> > There is _no_ container number in the non-container environment (or it
> > will be same for _all_ tasks).
>
> Chandra, virtual container number is essentially the same as user id
> in non-container environment. UBC were desgined for _users_ first.
> Containers were just the first environment which started to use it widely.

I am not denying any of the above :)

I think my original point is getting lost in the discussion, which is,
there should be way (for the sysadmin) to get a list of tasks belonging
to a resource group (in a non-container environment).
>
> And I really disagree when you say that non-container usecase is
> a superset of container usecase. I believe it is vice versa, since

I meant _only_ w.r.t resource management. My earlier replies were
pointing quite a few of those. here are a few:

- ability for the sysadmin to move a task to a resource group.
- assignment of task to a resource group should be transparent to the
  app.
- a resource group could exist with no tasks associated.

Containers can work without these features (and as OpenVZ proves it does
work). But, for a QoS type of resource management framework these are
mandatory.

> in container usecase you have a _full_ environment with root user and need
> more resources to be taken into account.

Support for different resources is a different topic. Users (of the two
models) can decide to control as many (or as few) resources as they
want. What I am talking here is about the ability of the framework.

>
> Thanks,
> Kirill
>
>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?

> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Alan Cox on Fri, 25 Aug 2006 20:25:31 GMT
View Forum Message <> Reply to Message

Ar Gwe, 2006-08-25 am 11:21 -0700, ysgrifennodd Chandra Seetharaman:
> But, the problem is that the struct user_beancounter (part of
> wombat_controller above) is a _copy_ of the original, not the original
> itself. We cannot keep the original (in _each_ controller), as there may
> be more than one controller in the system

Why would you want more than one controller for a given beancounter (and
thus a single measured resource). Can you give an example ?

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Alan Cox on Fri, 25 Aug 2006 20:32:30 GMT
View Forum Message <> Reply to Message

Ar Gwe, 2006-08-25 am 11:47 -0700, ysgrifennodd Chandra Seetharaman:
> I think my original point is getting lost in the discussion, which is,
> there should be way (for the sysadmin) to get a list of tasks belonging
> to a resource group (in a non-container environment).

Ok that much is easy to deal with. You print the luid in /proc.

> - ability for the sysadmin to move a task to a resource group.

So you want a setpluid(pid, luid) ? Trivial to add although you might
want to refuse it in many secure environments but thats an SELinux rule
again.

> - assignment of task to a resource group should be transparent to the
> app.

In those cases its akin to and matches security domain transitions which
says to me SELinux (or AppArmour) should do it.

> - a resource group could exist with no tasks associated.

Bean counters can exist with no tasks, and the CKRM people have been
corrected repeatedly on this point.

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Chandra Seetharaman on Fri, 25 Aug 2006 21:37:25 GMT
View Forum Message <> Reply to Message

On Fri, 2006-08-25 at 21:46 +0100, Alan Cox wrote:
> Ar Gwe, 2006-08-25 am 11:21 -0700, ysgrifennodd Chandra Seetharaman:
> > But, the problem is that the struct user_beancounter (part of
> > wombat_controller above) is a _copy_ of the original, not the original
> > itself. We cannot keep the original (in _each_ controller), as there may
> > be more than one controller in the system
>
> Why would you want more than one controller for a given beancounter (and
> thus a single measured resource). Can you give an example ?
>

Hmm... from what I see, data structure user_beancounter is _not_ defined
for a single resource, it is a beancounter for _all_ resources.

```
struct user_beancounter
{
 atomic_t  ub_refcount;
 spinlock_t  ub_lock;
 uid_t   ub_uid;
 struct hlist_node hash;

 struct user_beancounter *parent;
 void   *private_data;

 /* resources statistics and settings */
 struct ubparm  ub_parms[UB_RESOURCES];
};
```

ub_parms of _all_ controllers are held in this data structure.

So, keeping the beancounter data structure inside _a_ controller
specific data structure doesn't sound right to me, as other controllers
might also have the same need ?!

Controller _owns_ only ub_parms[controller_id], not the whole
user_beancounter, right ?

--

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

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Chandra Seetharaman on Fri, 25 Aug 2006 22:23:24 GMT
View Forum Message <> Reply to Message

On Fri, 2006-08-25 at 21:52 +0100, Alan Cox wrote:
> Ar Gwe, 2006-08-25 am 11:47 -0700, ysgrifennodd Chandra Seetharaman:
> > I think my original point is getting lost in the discussion, which is,
> > there should be way (for the sysadmin) to get a list of tasks belonging
> > to a resource group (in a non-container environment).
>
> Ok that much is easy to deal with. You print the luid in /proc.
>
> > - ability for the sysadmin to move a task to a resource group.
>
> So you want a setpluid(pid, luid) ? Trivial to add although you might
> want to refuse it in many secure environments but thats an SELinux rule
> again.

yes.
>
> > - assignment of task to a resource group should be transparent to the
> > app.
>
> In those cases its akin to and matches security domain transitions which
> says to me SELinux (or AppArmour) should do it.

If setpluid(pid, luid) exists, then this is more easy to handle.
>
> > - a resource group could exist with no tasks associated.
>
> Bean counters can exist with no tasks, and the CKRM people have been
> corrected repeatedly on this point.

Hmm... from what I understand from the code, when the last resource in
the beancounter is dropped, the beancounter is destroyed. Which to me
means that when there are no tasks in a beancounter it will be
destroyed. (I just tested the code and verified that the beancounter is

destroyed when the task dies).

Please correct me if my understanding is incorrect.

Let me reword the requirement: beancounter/resource group should _not_
be destroyed implicitly. It should be destroyed only when requested by
the user/sysadmin. In other words, we need a create_luid() and
destroy_luid().
>
>
>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Alan Cox on Fri, 25 Aug 2006 22:30:31 GMT
View Forum Message <> Reply to Message

Ar Gwe, 2006-08-25 am 14:37 -0700, ysgrifennodd Chandra Seetharaman:
> /* resources statistics and settings */
> struct ubparm ub_parms[UB_RESOURCES];
> };
>
> ub_parms of _all_ controllers are held in this data structure.
>
> So, keeping the beancounter data structure inside _a_ controller
> specific data structure doesn't sound right to me, as other controllers
> might also have the same need ?!
>
> Controller _owns_ only ub_parms[controller_id], not the whole
> user_beancounter, right ?

Right now I understand you

So you need

struct controller *ub_controller[UB_RESOURCES];

?

Alan

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Alan Cox on Fri, 25 Aug 2006 22:51:44 GMT
View Forum Message <> Reply to Message

Ar Gwe, 2006-08-25 am 15:23 -0700, ysgrifennodd Chandra Seetharaman:
> > Bean counters can exist with no tasks, and the CKRM people have been
> > corrected repeatedly on this point.
>
> Hmm... from what I understand from the code, when the last resource in
> the beancounter is dropped, the beancounter is destroyed. Which to me
> means that when there are no tasks in a beancounter it will be
> destroyed. (I just tested the code and verified that the beancounter is
> destroyed when the task dies).

If a task created resource remains then the beancounter remains until
the resources are destroyed, so it may exit well after the last task (eg
an object handed to another process with a different luid is stil
charged to us)

> Let me reword the requirement: beancounter/resource group should _not_
> be destroyed implicitly. It should be destroyed only when requested by
> the user/sysadmin. In other words, we need a create_luid() and
> destroy_luid().

So that you can preserve the limits on the resource group ? That also
makes sense if you are trying to do long term resource management.

Alan

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Chandra Seetharaman on Fri, 25 Aug 2006 22:59:26 GMT
View Forum Message <> Reply to Message

On Fri, 2006-08-25 at 23:51 +0100, Alan Cox wrote:
> Ar Gwe, 2006-08-25 am 14:37 -0700, ysgrifennodd Chandra Seetharaman:
> > /* resources statistics and settings */
> > struct ubparm ub_parms[UB_RESOURCES];
> > };

> >
> > ub_parms of _all_ controllers are held in this data structure.
> >
> > So, keeping the beancounter data structure inside _a_ controller
> > specific data structure doesn't sound right to me, as other controllers
> > might also have the same need ?!
> >
> > Controller _owns_ only ub_parms[controller_id], not the whole
> > user_beancounter, right ?
>
> Right now I understand you
>
> So you need
>
> struct controller *ub_controller[UB_RESOURCES];
>
> ?

exactly :)
>
> Alan
>
>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

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

Subject: Re: [ckrm-tech] [RFC][PATCH] UBC: user resource beancounters
Posted by Chandra Seetharaman on Fri, 25 Aug 2006 23:00:54 GMT
View Forum Message <> Reply to Message

On Sat, 2006-08-26 at 00:12 +0100, Alan Cox wrote:
> Ar Gwe, 2006-08-25 am 15:23 -0700, ysgrifennodd Chandra Seetharaman:
> > > Bean counters can exist with no tasks, and the CKRM people have been
> > > corrected repeatedly on this point.
> >

> > Hmm... from what I understand from the code, when the last resource in
> > the beancounter is dropped, the beancounter is destroyed. Which to me
> > means that when there are no tasks in a beancounter it will be
> > destroyed. (I just tested the code and verified that the beancounter is
> > destroyed when the task dies).
>
> If a task created resource remains then the beancounter remains until
> the resources are destroyed, so it may exit well after the last task (eg
> an object handed to another process with a different luid is stil
> charged to us)
>

It is the _implicit destruction_ that is a problem.

> > Let me reword the requirement: beancounter/resource group should _not_
> > be destroyed implicitly. It should be destroyed only when requested by
> > the user/sysadmin. In other words, we need a create_luid() and
> > destroy_luid().
>
> So that you can preserve the limits on the resource group ? That also
> makes sense if you are trying to do long term resource management.

Yup.

>
> Alan
--

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