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

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

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

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

```
 include/bc/task.h       |   59 +++++++++++++++++++++++++++++++++++++++++++++++++++++
 include/linux/sched.h   |    5 ++++
 kernel/bc/Makefile      |    1
 kernel/bc/beancounter.c |    3 ++
 kernel/bc/misc.c        |   32 ++++++++++++++++++++++++++
 kernel/fork.c           |   17 +++++++++++++-
 kernel/irq/handle.c     |    9 +++++++
 kernel/softirq.c        |    8 ++++++
 8 files changed, 132 insertions(+), 2 deletions(-)

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/task.h 2006-08-28 12:54:48.000000000 +0400
@@ -0,0 +1,59 @@
+/*
+ *  include/bc/task.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#ifndef __BC_TASK_H_
+#define __BC_TASK_H_
+
+struct beancounter;
+
+struct task_beancounter {
+ struct beancounter *exec_bc;
+ struct beancounter *fork_bc;
+};
+
```

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

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

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

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

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

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

```
+
+int bc_task_charge(struct task_struct *parent, struct task_struct *new)
+{
+ struct task_beancounter *old_bc;
+ struct task_beancounter *new_bc;
+ struct beancounter *bc;
+
+ old_bc = &parent->task_bc;
+ new_bc = &new->task_bc;
+
+ bc = old_bc->fork_bc;
+ new_bc->exec_bc = get_beancounter(bc);
+ new_bc->fork_bc = get_beancounter(bc);
+ return 0;
+}
+
+void bc_task_uncharge(struct task_struct *tsk)
+{
+ put_beancounter(tsk->task_bc.exec_bc);
+ put_beancounter(tsk->task_bc.fork_bc);
+}
--- ./kernel/fork.c.bctask 2006-08-28 12:20:13.000000000 +0400
+++ ./kernel/fork.c 2006-08-28 12:52:11.000000000 +0400
@@ -49,6 +49,8 @@
 #include <linux/taskstats_kern.h>
 #include <linux/random.h>

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

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

```
+}
 EXPORT_SYMBOL(free_task);

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

+ if (bc_task_charge(current, p))
+  goto bad_charge;
+
 #ifdef CONFIG_TRACE_IRQFLAGS
  DEBUG_LOCKS_WARN_ON(!p->hardirqs_enabled);
  DEBUG_LOCKS_WARN_ON(!p->softirqs_enabled);
@@ -1293,7 +1304,9 @@ bad_fork_cleanup_count:
  atomic_dec(&p->user->processes);
  free_uid(p->user);
 bad_fork_free:
- free_task(p);
+ bc_task_uncharge(p);
+bad_charge:
+ __free_task(p);
 fork_out:
  return ERR_PTR(retval);
 }
--- ./kernel/irq/handle.c.bctask 2006-07-10 12:39:20.000000000 +0400
+++ ./kernel/irq/handle.c 2006-08-28 12:52:11.000000000 +0400
@@ -16,6 +16,9 @@
 #include <linux/interrupt.h>
 #include <linux/kernel_stat.h>

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

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

  kstat_this_cpu.irqs[irq]++;
  if (CHECK_IRQ_PER_CPU(desc->status)) {
@@ -178,6 +184,8 @@ fastcall unsigned int __do_IRQ(unsigned
```

```
   desc->chip->ack(irq);
  action_ret = handle_IRQ_event(irq, regs, desc->action);
  desc->chip->end(irq);
+
+  reset_exec_bc(bc, &init_bc);
  return 1;
 }

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

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

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

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

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

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

#ifndef __ARCH_HAS_DO_SOFTIRQ
```