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);
> +}
>