Subject: Re: [PATCH 4/8] RSS container core
Posted by Andrew Morton on Wed, 30 May 2007 21:46:17 GMT
View Forum Message <> Reply to Message

On Wed, 30 May 2007 19:32:14 +0400
Pavel Emelianov <xemul@openvz.org> wrote:

> The core routines for tracking the page ownership,
> registration of RSS subsystem in the containers and
> the definition of rss_container as container subsystem
> combined with resource counter.
>
> --- linux-2.6.22-rc2-mm1.orig/include/linux/rss_container.h 2007-05-30 16:16:58.000000000
+0400
> +++ linux-2.6.22-rc2-mm1-0/include/linux/rss_container.h 2007-05-30 16:13:09.000000000
+0400
> @@ -0,0 +1,64 @@
> +#ifndef __RSS_CONTAINER_H__
> +#define __RSS_CONTAINER_H__
> +/*
> + * RSS container
> + *
> + * Copyright 2007 OpenVZ SWsoft Inc
> + *
> + * Author: Pavel Emelianov <xemul@openvz.org>
> + *
> + */
> +
> +struct page_container;
> +struct rss_container;
> +
> +#ifdef CONFIG_RSS_CONTAINER
> +int container_rss_prepare(struct page *, struct vm_area_struct *vma,
> + struct page_container **);
> +
> +void container_rss_add(struct page_container *);
> +void container_rss_del(struct page_container *);
> +void container_rss_release(struct page_container *);
> +
> +void mm_init_container(struct mm_struct *mm, struct task_struct *tsk);
> +void mm_free_container(struct mm_struct *mm);
> +
> +static inline void init_page_container(struct page *page)
> +{
> + page_container(page) = NULL;
> +}

Please, no. Using a "function" as an lvalue like this _requires_ that the

"function" be implemented as a macro, which rather defeats the whole point.

Much nicer to do

```
#ifdef CONFIG_FOO
static inline void set_page_container(struct page *page,
     struct page_container *page_container)
{
 page->rss_container = page_container;
}
#else
static inline void set_page_container(struct page *page,
     struct page_container *page_container)
{
}
#endif
```

> +#else
> +static inline int container_rss_prepare(struct page *pg,
> + struct vm_area_struct *vma, struct page_container **pc)
> +{
> + *pc = NULL; /* to make gcc happy */

eh? What is gcc's problem here?

> + return 0;
> +}
>
> ...
>
> @@ -0,0 +1,271 @@
> +/*
> + * RSS accounting container
> + *
> + * Copyright 2007 OpenVZ SWsoft Inc
> + *
> + * Author: Pavel Emelianov <xemul@openvz.org>
> + *
> + */
> +
> +#include <linux/list.h>
> +#include <linux/sched.h>
> +#include <linux/mm.h>
> +#include <linux/swap.h>
> +#include <linux/res_counter.h>
> +#include <linux/rss_container.h>
> +
> +struct rss_container {

> + struct res_counter res;
> + struct list_head inactive_list;
> + struct list_head active_list;
> + atomic_t rss_reclaimed;
> + struct container_subsys_state css;
> +};
> +
> +struct page_container {
> + struct page *page;
> + struct rss_container *cnt;
> + struct list_head list;
> +};

Document each member carefully, please. This is where your readers will
first come when trying to understand your code.

> +}
> +
> +int container_rss_prepare(struct page *page, struct vm_area_struct *vma,
> + struct page_container **ppc)
> +{

This is an important-looking kernel-wide function. It needs a nice comment
telling people what its role in life is.

> + struct rss_container *rss;
> + struct page_container *pc;
> +
> + rcu_read_lock();
> + rss = rcu_dereference(vma->vm_mm->rss_container);
> + css_get(&rss->css);
> + rcu_read_unlock();
> +
> + pc = kmalloc(sizeof(struct page_container), GFP_KERNEL);
> + if (pc == NULL)
> + goto out_nomem;
> +
> + while (res_counter_charge(&rss->res, 1)) {
> + if (try_to_free_pages_in_container(rss)) {
> +   atomic_inc(&rss->rss_reclaimed);
> +   continue;
> + }

I find it mysterious that rss->rss_reclaimed gets incremented by one when
try_to_free_pages_in_container() succeeds (or is it when it fails?
try_to_free_pages_in_container() is undocumented, so it's hard to tell).
rss_reclaimed is also undocumented, adding to the mystery.

Also, try_to_free_pages_in_container() gets added in a later patch. The whole series seems backwards. Probably this problem is less serious, as things will still build and run when CONFIG_RSS_CONTAINER=n

```
> +  container_out_of_memory(rss);
> +  if (test_thread_flag(TIF_MEMDIE))
> +   goto out_charge;
> + }
> +
> + pc->page = page;
> + pc->cnt = rss;
> + *ppc = pc;
> + return 0;
> +
> +out_charge:
> + kfree(pc);
> +out_nomem:
> + css_put(&rss->css);
> + return -ENOMEM;
> +}
> +
> +void container_rss_release(struct page_container *pc)
> +{
> + struct rss_container *rss;
> +
> + rss = pc->cnt;
> + res_counter_uncharge(&rss->res, 1);
> + css_put(&rss->css);
> + kfree(pc);
> +}
> +
> +void container_rss_add(struct page_container *pc)
> +{
> + struct page *pg;
> + struct rss_container *rss;
> +
> + pg = pc->page;
> + rss = pc->cnt;
> +
> + spin_lock_irq(&rss->res.lock);
> + list_add(&pc->list, &rss->active_list);
> + spin_unlock_irq(&rss->res.lock);
> +
> + page_container(pg) = pc;
> +}
> +
> +void container_rss_del(struct page_container *pc)
```

```
> +{
> + struct rss_container *rss;
> +
> + rss = pc->cnt;
> + spin_lock_irq(&rss->res.lock);
> + list_del(&pc->list);
> + res_counter_uncharge_locked(&rss->res, 1);
> + spin_unlock_irq(&rss->res.lock);
> +
> + css_put(&rss->css);
> + kfree(pc);
> +}
```

Please document all the above.

```
> +static void rss_move_task(struct container_subsys *ss,
> +  struct container *cont,
> +  struct container *old_cont,
> +  struct task_struct *p)
> +{
> + struct mm_struct *mm;
> + struct rss_container *rss, *old_rss;
> +
> + mm = get_task_mm(p);
> + if (mm == NULL)
> +  goto out;
> +
> + rss = rss_from_cont(cont);
> + old_rss = rss_from_cont(old_cont);
> + if (old_rss != mm->rss_container)
> +  goto out_put;
```

I cannot tell what would cause the above test-and-goto to be taken.
Please add a comment describing this piece of code.

```
> + css_get(&rss->css);
> + rcu_assign_pointer(mm->rss_container, rss);
> + css_put(&old_rss->css);
> +
> +out_put:
> + mmput(mm);
> +out:
> + return;
> +}
> +
> +static struct rss_container init_rss_container;
> +
> +
```

```
> +static ssize_t rss_read(struct container *cont, struct cftype *cft,
> +  struct file *file, char __user *userbuf,
> +  size_t nbytes, loff_t *ppos)
> +{
> + return res_counter_read(&rss_from_cont(cont)->res, cft->private,
> +   userbuf, nbytes, ppos);
> +}
> +
> +static ssize_t rss_write(struct container *cont, struct cftype *cft,
> +  struct file *file, const char __user *userbuf,
> +  size_t nbytes, loff_t *ppos)
> +{
> + return res_counter_write(&rss_from_cont(cont)->res, cft->private,
> +   userbuf, nbytes, ppos);
> +}
> +
> +static ssize_t rss_read_reclaimed(struct container *cont, struct cftype *cft,
> +  struct file *file, char __user *userbuf,
> +  size_t nbytes, loff_t *ppos)
> +{
> + char buf[64], *s;
> +
> + s = buf;
> + s += sprintf(s, "%d\n",
> +   atomic_read(&rss_from_cont(cont)->rss_reclaimed));
> + return simple_read_from_buffer((void __user *)userbuf, nbytes,
> +   ppos, buf, s - buf);
> +}
> +
> +
> +static struct cftype rss_usage = {
> + .name = "rss_usage",
> + .private = RES_USAGE,
> + .read = rss_read,
> +};
> +
> +static struct cftype rss_limit = {
> + .name = "rss_limit",
> + .private = RES_LIMIT,
> + .read = rss_read,
> + .write = rss_write,
> +};
> +
> +static struct cftype rss_failcnt = {
> + .name = "rss_failcnt",
> + .private = RES_FAILCNT,
> + .read = rss_read,
> +};
```

> +
> +static struct cftype rss_reclaimed = {
> + .name = "rss_reclaimed",
> + .read = rss_read_reclaimed,
> +};

Did we add user documentation for the above?

> +static int rss_populate(struct container_subsys *ss,
> + struct container *cont)
> +{
> + int rc;
> +
> + if ((rc = container_add_file(cont, &rss_usage)) < 0)
> + return rc;
> + if ((rc = container_add_file(cont, &rss_failcnt)) < 0)
> + return rc;
> + if ((rc = container_add_file(cont, &rss_limit)) < 0)
> + return rc;
> + if ((rc = container_add_file(cont, &rss_reclaimed)) < 0)
> + return rc;

If we fail partway through here, do the thus-far-created fiels get cleaned up?

> + return 0;
> +}
> +
> +struct container_subsys rss_subsys = {
> + .name = "rss",
> + .subsys_id = rss_subsys_id,
> + .create = rss_create,
> + .destroy = rss_destroy,
> + .populate = rss_populate,
> + .attach = rss_move_task,
> + .early_init = 1,
> +};

Did this need kernel-wide scope?