Subject: [PATCH 4/8] RSS container core
Posted by xemul on Mon, 09 Apr 2007 12:45:19 GMT
View Forum Message <> Reply to Message

This includes
* definition of rss_container as container subsystem combined
  with resource counter;
* registration of RSS container in generic containers;
* routines for pages tracking.

```
diff -upr linux-2.6.20.orig/include/linux/container_subsys.h
linux-2.6.20-2/include/linux/container_subsys.h
--- linux-2.6.20.orig/include/linux/container_subsys.h 2007-04-09 11:26:06.000000000 +0400
+++ linux-2.6.20-2/include/linux/container_subsys.h 2007-04-09 11:26:06.000000000 +0400
@@ -9,6 +9,10 @@
 SUBSYS(cpuset)
 #endif

+#ifdef CONFIG_RSS_CONTAINER
+SUBSYS(rss)
+#endif
+
 /* */

 /* */
diff -upr linux-2.6.20.orig/include/linux/rss_container.h linux-2.6.20-2/include/linux/rss_container.h
--- linux-2.6.20.orig/include/linux/rss_container.h 2007-04-09 11:26:12.000000000 +0400
+++ linux-2.6.20-2/include/linux/rss_container.h 2007-04-09 11:26:06.000000000 +0400
@@ -0,0 +1,55 @@
+#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);
+
+#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 */
+ return 0;
+}
+
+static inline void container_rss_add(struct page_container *pc)
+{
+}
+
+static inline void container_rss_del(struct page_container *pc)
+{
+}
+
+static inline void container_rss_release(struct page_container *pc)
+{
+}
+
+static inline void mm_init_container(struct mm_struct *mm, struct task_struct *t)
+{
+}
+
+static inline void mm_free_container(struct mm_struct *mm)
+{
+}
+
+#endif
+#endif
diff -upr linux-2.6.20.orig/init/Kconfig linux-2.6.20-2/init/Kconfig
--- linux-2.6.20.orig/init/Kconfig 2007-04-09 11:26:06.000000000 +0400
+++ linux-2.6.20-2/init/Kconfig 2007-04-09 11:26:06.000000000 +0400
@@ -257,6 +257,17 @@ config CPUSETS
  bool
  select CONTAINERS

+config RSS_CONTAINER
+ bool "RSS accounting container"
+ select RESOURCE_COUNTERS
+ help
+   Provides a simple Resource Controller for monitoring and
```

```
+   controlling the total Resident Set Size of the tasks in a container
+   The reclaim logic is now container aware, when the container goes
+   overlimit the page reclaimer reclaims pages belonging to this
+   container. If we are unable to reclaim enough pages to satisfy the
+   request, the process is killed with an out of memory warning.
+
 config SYSFS_DEPRECATED
  bool "Create deprecated sysfs files"
  default y
diff -upr linux-2.6.20.orig/mm/Makefile linux-2.6.20-2/mm/Makefile
--- linux-2.6.20.orig/mm/Makefile 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/mm/Makefile 2007-04-09 11:26:06.000000000 +0400
@@ -29,3 +29,5 @@ obj-$(CONFIG_MEMORY_HOTPLUG) += memory_h
 obj-$(CONFIG_FS_XIP) += filemap_xip.o
 obj-$(CONFIG_MIGRATION) += migrate.o
 obj-$(CONFIG_SMP) += allocpercpu.o
+
+obj-$(CONFIG_RSS_CONTAINER) += rss_container.o
diff -upr linux-2.6.20.orig/mm/rss_container.c linux-2.6.20-2/mm/rss_container.c
--- linux-2.6.20.orig/mm/rss_container.c 2007-04-09 11:26:12.000000000 +0400
+++ linux-2.6.20-2/mm/rss_container.c 2007-04-09 11:26:06.000000000 +0400
@@ -0,0 +1,274 @@
+/*
+ * 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;
+};
+
+static inline struct rss_container *rss_from_cont(struct container *cnt)
+{
+ return container_of(container_subsys_state(cnt, rss_subsys_id),
+   struct rss_container, css);
+}
+
+void mm_init_container(struct mm_struct *mm, struct task_struct *tsk)
+{
+ struct rss_container *cnt;
+
+ cnt = rss_from_cont(task_container(tsk, rss_subsys_id));
+ css_get(&cnt->css);
+ mm->rss_container = cnt;
+}
+
+void mm_free_container(struct mm_struct *mm)
+{
+ css_put(&mm->rss_container->css);
+}
+
+int container_rss_prepare(struct page *page, struct vm_area_struct *vma,
+  struct page_container **ppc)
+{
+ 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;
+  }
+
+  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 page *page;
+ struct rss_container *rss;
+
+ page = pc->page;
+ 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);
+}
+
+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;
+
+ 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 inline void rss_container_attach(struct rss_container *rss,
+  struct container *cont)
+{
+ cont->subsys[rss_subsys_id] = &rss->css;
+ rss->css.container = cont;
+}
+
+static int rss_create(struct container_subsys *ss, struct container *cont)
+{
+ struct rss_container *rss;
+
```

```
+ if (unlikely(cont->parent == NULL)) {
+  rss = &init_rss_container;
+  css_get(&rss->css);
+  init_mm.rss_container = rss;
+ } else
+  rss = kzalloc(sizeof(struct rss_container), GFP_KERNEL);
+
+ if (rss == NULL)
+  return -ENOMEM;
+
+ res_counter_init(&rss->res);
+ INIT_LIST_HEAD(&rss->inactive_list);
+ INIT_LIST_HEAD(&rss->active_list);
+ rss_container_attach(rss, cont);
+ return 0;
+}
+
+static void rss_destroy(struct container_subsys *ss,
+  struct container *cont)
+{
+ kfree(rss_from_cont(cont));
+}
+
+
+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,
+};
+
+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;
+
+ 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,
+};
```