Subject: [RFC][PATCH][1/4] RSS controller setup
Posted by Balbir Singh on Mon, 19 Feb 2007 06:50:26 GMT
View Forum Message <> Reply to Message

This patch sets up the basic controller infrastructure on top of the
containers infrastructure. Two files are provided for monitoring
and control memctlr_usage and memctlr_limit.

memctlr_usage shows the current usage (in pages, of RSS) and the limit
set by the user.

memctlr_limit can be used to set a limit on the RSS usage of the resource.
A special value of 0, indicates that the usage is unlimited. The limit
is set in units of pages.

Signed-off-by: <balbir@in.ibm.com>

```
 include/linux/memctlr.h |   22 ++++++
 init/Kconfig            |    7 +
 mm/Makefile             |    1
 mm/memctlr.c            |  169 ++++++++++++++++++++++++++++++++++++++++++++++++
 4 files changed, 199 insertions(+)

diff -puN /dev/null include/linux/memctlr.h
--- /dev/null 2007-02-02 22:51:23.000000000 +0530
+++ linux-2.6.20-balbir/include/linux/memctlr.h 2007-02-16 00:22:11.000000000 +0530
@@ -0,0 +1,22 @@
+/* memctlr.h - Memory Controller for containers
+ *
+ * Copyright (C) Balbir Singh,   IBM Corp. 2006-2007
+ *
+ * This program is free software; you can redistribute it and/or modify it
+ * under the terms of version 2.1 of the GNU Lesser General Public License
+ * as published by the Free Software Foundation.
+ *
+ * This program is distributed in the hope that it would be useful, but
+ * WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
+ */
+
+#ifndef _LINUX_MEMCTLR_H
+#define _LINUX_MEMCTLR_H
+
+#ifdef CONFIG_CONTAINER_MEMCTLR
+
+#else /* CONFIG_CONTAINER_MEMCTLR  */
```

```
+
+#endif /* CONFIG_CONTAINER_MEMCTLR */
+#endif /* _LINUX_MEMCTLR_H */
diff -puN init/Kconfig~memctlr-setup init/Kconfig
--- linux-2.6.20/init/Kconfig~memctlr-setup 2007-02-15 21:58:42.000000000 +0530
+++ linux-2.6.20-balbir/init/Kconfig 2007-02-15 21:58:42.000000000 +0530
@@ -306,6 +306,13 @@ config CONTAINER_NS
          for instance virtual servers and checkpoint/restart
          jobs.

+config CONTAINER_MEMCTLR
+ bool "A simple RSS based memory controller"
+ select CONTAINERS
+ help
+   Provides a simple Resource Controller for monitoring and
+   controlling the total Resident Set Size of the tasks in a container
+
 config RELAY
  bool "Kernel->user space relay support (formerly relayfs)"
  help
diff -puN mm/Makefile~memctlr-setup mm/Makefile
--- linux-2.6.20/mm/Makefile~memctlr-setup 2007-02-15 21:58:42.000000000 +0530
+++ linux-2.6.20-balbir/mm/Makefile 2007-02-15 21:58:42.000000000 +0530
@@ -29,3 +29,4 @@ 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_CONTAINER_MEMCTLR) += memctlr.o
diff -puN /dev/null mm/memctlr.c
--- /dev/null 2007-02-02 22:51:23.000000000 +0530
+++ linux-2.6.20-balbir/mm/memctlr.c 2007-02-16 00:22:11.000000000 +0530
@@ -0,0 +1,169 @@
+/*
+ * memctlr.c - Memory Controller for containers
+ *
+ * Copyright (C) Balbir Singh,   IBM Corp. 2006-2007
+ *
+ * This program is free software; you can redistribute it and/or modify it
+ * under the terms of version 2.1 of the GNU Lesser General Public License
+ * as published by the Free Software Foundation.
+ *
+ * This program is distributed in the hope that it would be useful, but
+ * WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
+ */
+
+#include <linux/init.h>
+#include <linux/parser.h>
```

```
+#include <linux/fs.h>
+#include <linux/container.h>
+#include <linux/memctlr.h>
+
+#include <asm/uaccess.h>
+
+#define RES_USAGE_NO_LIMIT   0
+static const char version[] = "0.1";
+
+struct res_counter {
+ unsigned long usage; /* The current usage of the resource being */
+    /* counted        */
+ unsigned long limit; /* The limit on the resource     */
+ unsigned long nr_limit_exceeded;
+};
+
+struct memctlr {
+ struct container_subsys_state css;
+ struct res_counter   counter;
+ spinlock_t    lock;
+};
+
+static struct container_subsys memctlr_subsys;
+
+static inline struct memctlr *memctlr_from_cont(struct container *cont)
+{
+ return container_of(container_subsys_state(cont, &memctlr_subsys),
+    struct memctlr, css);
+}
+
+static inline struct memctlr *memctlr_from_task(struct task_struct *p)
+{
+ return memctlr_from_cont(task_container(p, &memctlr_subsys));
+}
+
+static int memctlr_create(struct container_subsys *ss, struct container *cont)
+{
+ struct memctlr *mem = kzalloc(sizeof(*mem), GFP_KERNEL);
+ if (!mem)
+  return -ENOMEM;
+
+ spin_lock_init(&mem->lock);
+ cont->subsys[memctlr_subsys.subsys_id] = &mem->css;
+ return 0;
+}
+
+static void memctlr_destroy(struct container_subsys *ss,
+    struct container *cont)
```

```
+{
+ kfree(memctlr_from_cont(cont));
+}
+
+static ssize_t memctlr_write(struct container *cont, struct cftype *cft,
+    struct file *file, const char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
+ char *buffer;
+ int ret = 0;
+ unsigned long limit;
+ struct memctlr *mem = memctlr_from_cont(cont);
+
+ BUG_ON(!mem);
+ if ((buffer = kmalloc(nbytes + 1, GFP_KERNEL)) == 0)
+  return -ENOMEM;
+
+ buffer[nbytes] = 0;
+ if (copy_from_user(buffer, userbuf, nbytes)) {
+  ret = -EFAULT;
+  goto out_err;
+ }
+
+ container_manage_lock();
+ if (container_is_removed(cont)) {
+  ret = -ENODEV;
+  goto out_unlock;
+ }
+
+ limit = simple_strtoul(buffer, NULL, 10);
+ /*
+  * 0 is a valid limit (unlimited resource usage)
+  */
+ if (!limit && strcmp(buffer, "0"))
+  goto out_unlock;
+
+ spin_lock(&mem->lock);
+ mem->counter.limit = limit;
+ spin_unlock(&mem->lock);
+
+ ret = nbytes;
+out_unlock:
+ container_manage_unlock();
+out_err:
+ kfree(buffer);
+ return ret;
+}
+
+static ssize_t memctlr_read(struct container *cont, struct cftype *cft,
+    struct file *file, char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
+ unsigned long usage, limit;
+ char usagebuf[64];  /* Move away from stack later */
+ char *s = usagebuf;
+ struct memctlr *mem = memctlr_from_cont(cont);
+
+ spin_lock(&mem->lock);
+ usage = mem->counter.usage;
+ limit = mem->counter.limit;
+ spin_unlock(&mem->lock);
+
+ s += sprintf(s, "usage %lu, limit %ld\n", usage, limit);
+ return simple_read_from_buffer(userbuf, nbytes, ppos, usagebuf,
+     s - usagebuf);
+}
+
+static struct cftype memctlr_usage = {
+ .name = "memctlr_usage",
+ .read = memctlr_read,
+};
+
+static struct cftype memctlr_limit = {
+ .name = "memctlr_limit",
+ .write = memctlr_write,
+};
+
+static int memctlr_populate(struct container_subsys *ss,
+    struct container *cont)
+{
+ int rc;
+ if ((rc = container_add_file(cont, &memctlr_usage)) < 0)
+  return rc;
+ if ((rc = container_add_file(cont, &memctlr_limit)) < 0)
+  return rc;
+ return 0;
+}
+
+static struct container_subsys memctlr_subsys = {
+ .name = "memctlr",
+ .create = memctlr_create,
+ .destroy = memctlr_destroy,
+ .populate = memctlr_populate,
+};
+
+int __init memctlr_init(void)
```

```
+{
+ int id;
+
+ id = container_register_subsys(&memctlr_subsys);
+ printk("Initializing memctlr version %s, id %d\n", version, id);
+ return id < 0 ? id : 0;
+}
+
+module_init(memctlr_init);
```

–

--
Warm Regards,
Balbir Singh