Subject: [PATCH 2/2] cpu_accounting controller
Posted by Srivatsa Vaddagiri on Thu, 01 Mar 2007 13:50:08 GMT
View Forum Message <> Reply to Message

This patch demonstrates how a resource controller can work with rcfs.

The controller counts the total CPU time used by all processes in a
resource container, during the time that they're members of the
container.

Written by Paul Menage. Adapted to work with rcfs by Srivatsa.

Signed-off-by : Paul Menage <menage@google.com>
Signed-off-by : Srivatsa Vaddagiri <vatsa@in.ibm.com>

```
diff -puN /dev/null include/linux/cpu_acct.h
--- /dev/null 2006-02-25 03:06:56.000000000 +0530
+++ linux-2.6.20-vatsa/include/linux/cpu_acct.h 2007-03-01 16:53:39.000000000 +0530
@@ -0,0 +1,14 @@
+
+#ifndef _LINUX_CPU_ACCT_H
+#define _LINUX_CPU_ACCT_H
+
+#include <linux/rcfs.h>
+#include <asm/cputime.h>
+
+#ifdef CONFIG_RC_CPUACCT
+extern void cpuacct_charge(struct task_struct *, cputime_t cputime);
+#else
+static void inline cpuacct_charge(struct task_struct *p, cputime_t cputime) {}
+#endif
+
+#endif
diff -puN init/Kconfig~cpu_acct init/Kconfig
--- linux-2.6.20/init/Kconfig~cpu_acct 2007-03-01 16:53:39.000000000 +0530
+++ linux-2.6.20-vatsa/init/Kconfig 2007-03-01 16:53:39.000000000 +0530
@@ -291,6 +291,13 @@ config SYSFS_DEPRECATED
   If you are using a distro that was released in 2006 or later,
   it should be safe to say N here.

+config RC_CPUACCT
+ bool "Simple CPU accounting container subsystem"
+ select RCFS
+ help
```

```
+   Provides a simple Resource Controller for monitoring the
+   total CPU consumed by the tasks in a container
+
 config RELAY
  bool "Kernel->user space relay support (formerly relayfs)"
  help
diff -puN /dev/null kernel/cpu_acct.c
--- /dev/null 2006-02-25 03:06:56.000000000 +0530
+++ linux-2.6.20-vatsa/kernel/cpu_acct.c 2007-03-01 16:53:39.000000000 +0530
@@ -0,0 +1,221 @@
+/*
+ * kernel/cpu_acct.c - CPU accounting container subsystem
+ *
+ * Copyright (C) Google Inc, 2006
+ *
+ * Developed by Paul Menage (menage@google.com) and Balbir Singh
+ * (balbir@in.ibm.com)
+ *
+ */
+
+/*
+ * Container subsystem for reporting total CPU usage of tasks in a
+ * container, along with percentage load over a time interval
+ */
+
+#include <linux/module.h>
+#include <linux/nsproxy.h>
+#include <linux/rcfs.h>
+#include <linux/fs.h>
+#include <asm/div64.h>
+
+struct cpuacct {
+ spinlock_t lock;
+ /* total time used by this class */
+ cputime64_t time;
+
+ /* time when next load calculation occurs */
+ u64 next_interval_check;
+
+ /* time used in current period */
+ cputime64_t current_interval_time;
+
+ /* time used in last period */
+ cputime64_t last_interval_time;
+};
+
+static struct rc_subsys cpuacct_subsys;
+
```

```
+static inline struct cpuacct *nsproxy_ca(struct nsproxy *ns)
+{
+ if (!ns)
+  return NULL;
+
+ return ns->ctlr_data[cpuacct_subsys.subsys_id];
+}
+
+static inline struct cpuacct *task_ca(struct task_struct *task)
+{
+ return nsproxy_ca(task->nsproxy);
+}
+
+#define INTERVAL (HZ * 10)
+
+static inline u64 next_interval_boundary(u64 now) {
+ /* calculate the next interval boundary beyond the
+  * current time */
+ do_div(now, INTERVAL);
+ return (now + 1) * INTERVAL;
+}
+
+static int cpuacct_create(struct rc_subsys *ss, struct nsproxy *ns,
+      struct nsproxy *parent)
+{
+ struct cpuacct *ca;
+
+ if (parent && (parent != &init_nsproxy))
+  return -EINVAL;
+
+ ca = kzalloc(sizeof(*ca), GFP_KERNEL);
+ if (!ca)
+  return -ENOMEM;
+ spin_lock_init(&ca->lock);
+ ca->next_interval_check = next_interval_boundary(get_jiffies_64());
+ ns->ctlr_data[cpuacct_subsys.subsys_id] = ca;
+ return 0;
+}
+
+static void cpuacct_destroy(struct rc_subsys *ss, struct nsproxy *ns)
+{
+ kfree(nsproxy_ca(ns));
+}
+
+/* Lazily update the load calculation if necessary. Called with ca locked */
+static void cpuusage_update(struct cpuacct *ca)
+{
+ u64 now = get_jiffies_64();
```

```
+ /* If we're not due for an update, return */
+ if (ca->next_interval_check > now)
+  return;
+
+ if (ca->next_interval_check <= (now - INTERVAL)) {
+  /* If it's been more than an interval since the last
+   * check, then catch up - the last interval must have
+   * been zero load */
+  ca->last_interval_time = 0;
+  ca->next_interval_check = next_interval_boundary(now);
+ } else {
+  /* If a steal takes the last interval time negative,
+   * then we just ignore it */
+  if ((s64)ca->current_interval_time > 0) {
+   ca->last_interval_time = ca->current_interval_time;
+  } else {
+   ca->last_interval_time = 0;
+  }
+  ca->next_interval_check += INTERVAL;
+ }
+ ca->current_interval_time = 0;
+}
+
+static ssize_t cpuusage_read(struct nsproxy *ns,
+        struct cftype *cft,
+        struct file *file,
+        char __user *buf,
+        size_t nbytes, loff_t *ppos)
+{
+ struct cpuacct *ca = nsproxy_ca(ns);
+ u64 time;
+ char usagebuf[64];
+ char *s = usagebuf;
+
+ spin_lock_irq(&ca->lock);
+ cpuusage_update(ca);
+ time = cputime64_to_jiffies64(ca->time);
+ spin_unlock_irq(&ca->lock);
+
+ /* Convert 64-bit jiffies to seconds */
+ time *= 1000;
+ do_div(time, HZ);
+ s += sprintf(s, "%llu", (unsigned long long) time);
+
+ return simple_read_from_buffer(buf, nbytes, ppos, usagebuf, s - usagebuf);
+}
+
+static ssize_t load_read(struct nsproxy *ns,
```

```
+    struct cftype *cft,
+    struct file *file,
+    char __user *buf,
+    size_t nbytes, loff_t *ppos)
+{
+ struct cpuacct *ca = nsproxy_ca(ns);
+ u64 time;
+ char usagebuf[64];
+ char *s = usagebuf;
+
+ /* Find the time used in the previous interval */
+ spin_lock_irq(&ca->lock);
+ cpuusage_update(ca);
+ time = cputime64_to_jiffies64(ca->last_interval_time);
+ spin_unlock_irq(&ca->lock);
+
+ /* Convert time to a percentage, to give the load in the
+  * previous period */
+ time *= 100;
+ do_div(time, INTERVAL);
+
+ s += sprintf(s, "%llu", (unsigned long long) time);
+
+ return simple_read_from_buffer(buf, nbytes, ppos, usagebuf, s - usagebuf);
+}
+
+static struct cftype cft_usage = {
+ .name = "cpuacct.usage",
+ .read = cpuusage_read,
+};
+
+static struct cftype cft_load = {
+ .name = "cpuacct.load",
+ .read = load_read,
+};
+
+static int cpuacct_populate(struct rc_subsys *ss,
+       struct dentry *d)
+{
+ int err;
+
+ if ((err = rcfs_add_file(d, &cft_usage)))
+  return err;
+ if ((err = rcfs_add_file(d, &cft_load)))
+  return err;
+
+ return 0;
+}
```

```
+
+
+void cpuacct_charge(struct task_struct *task, cputime_t cputime)
+{
+
+ struct cpuacct *ca;
+ unsigned long flags;
+
+ if (!cpuacct_subsys.active)
+  return;
+ rcu_read_lock();
+ ca = task_ca(task);
+ if (ca) {
+  spin_lock_irqsave(&ca->lock, flags);
+  cpuusage_update(ca);
+  ca->time = cputime64_add(ca->time, cputime);
+  ca->current_interval_time =
+   cputime64_add(ca->current_interval_time, cputime);
+  spin_unlock_irqrestore(&ca->lock, flags);
+ }
+ rcu_read_unlock();
+}
+
+static struct rc_subsys cpuacct_subsys = {
+ .name = "cpuacct",
+ .create = cpuacct_create,
+ .destroy = cpuacct_destroy,
+ .populate = cpuacct_populate,
+ .subsys_id = -1,
+};
+
+
+int __init init_cpuacct(void)
+{
+ int id = rc_register_subsys(&cpuacct_subsys);
+ return id < 0 ? id : 0;
+}
+
+module_init(init_cpuacct)
diff -puN kernel/Makefile~cpu_acct kernel/Makefile
--- linux-2.6.20/kernel/Makefile~cpu_acct 2007-03-01 16:53:39.000000000 +0530
+++ linux-2.6.20-vatsa/kernel/Makefile 2007-03-01 16:53:39.000000000 +0530
@@ -36,6 +36,7 @@ obj-$(CONFIG_BSD_PROCESS_ACCT) += acct.o
 obj-$(CONFIG_KEXEC) += kexec.o
 obj-$(CONFIG_COMPAT) += compat.o
 obj-$(CONFIG_CPUSETS) += cpuset.o
+obj-$(CONFIG_RC_CPUACCT) += cpu_acct.o
 obj-$(CONFIG_IKCONFIG) += configs.o
```

```
 obj-$(CONFIG_STOP_MACHINE) += stop_machine.o
 obj-$(CONFIG_AUDIT) += audit.o auditfilter.o
diff -puN kernel/sched.c~cpu_acct kernel/sched.c
--- linux-2.6.20/kernel/sched.c~cpu_acct 2007-03-01 16:53:39.000000000 +0530
+++ linux-2.6.20-vatsa/kernel/sched.c 2007-03-01 16:53:39.000000000 +0530
@@ -52,6 +52,7 @@
 #include <linux/tsacct_kern.h>
 #include <linux/kprobes.h>
 #include <linux/delayacct.h>
+#include <linux/cpu_acct.h>
 #include <asm/tlb.h>

 #include <asm/unistd.h>
@@ -3066,9 +3067,13 @@ void account_user_time(struct task_struc
 {
  struct cpu_usage_stat *cpustat = &kstat_this_cpu.cpustat;
  cputime64_t tmp;
+ struct rq *rq = this_rq();

  p->utime = cputime_add(p->utime, cputime);

+ if (p != rq->idle)
+  cpuacct_charge(p, cputime);
+
  /* Add user time to cpustat. */
  tmp = cputime_to_cputime64(cputime);
  if (TASK_NICE(p) > 0)
@@ -3098,9 +3103,10 @@ void account_system_time(struct task_str
   cpustat->irq = cputime64_add(cpustat->irq, tmp);
  else if (softirq_count())
   cpustat->softirq = cputime64_add(cpustat->softirq, tmp);
- else if (p != rq->idle)
+ else if (p != rq->idle) {
   cpustat->system = cputime64_add(cpustat->system, tmp);
- else if (atomic_read(&rq->nr_iowait) > 0)
+  cpuacct_charge(p, cputime);
+ } else if (atomic_read(&rq->nr_iowait) > 0)
   cpustat->iowait = cputime64_add(cpustat->iowait, tmp);
  else
   cpustat->idle = cputime64_add(cpustat->idle, tmp);
@@ -3125,8 +3131,10 @@ void account_steal_time(struct task_stru
    cpustat->iowait = cputime64_add(cpustat->iowait, tmp);
   else
    cpustat->idle = cputime64_add(cpustat->idle, tmp);
- } else
+ } else {
   cpustat->steal = cputime64_add(cpustat->steal, tmp);
+  cpuacct_charge(p, -tmp);
```

+ }
 }

 static void task_running_tick(struct rq *rq, struct task_struct *p)

_

--
Regards,
vatsa

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers