Subject: [PATCH 2/2] Hook up to (process) container feature in mm tree
Posted by Srivatsa Vaddagiri on Sat, 23 Jun 2007 13:20:04 GMT
View Forum Message <> Reply to Message

This patch hooks up cpu scheduler with Paul Menage's container infrastructure.

The container patches allows administrator to create arbitrary groups of tasks
and define resource allocation for each group. By registering with container
infrastructure, cpu scheduler is made aware of group membership information for
each task, creation/deletion of groups etc and can use that information to
provide fairness between groups.

This mechanism can indirectly be used to provide fairness between users
also. All that is needed is a user-space program (which is being working
upon) which monitors for PROC_EVENT_UID events (using process event connector)
and moves the task to appropriate user-directory in container filesystem.

As an example for "HOWTO use this feature", follow these steps:

1. Define CONFIG_FAIR_GROUP_SCHED (General Setup->Fair Group Scheduler)
   and compile the kernel
2. After booting:

```
# cd /dev
# mkdir cpuctl
# mount -t container -ocpuctl none /dev/cpuctl
# cd cpuctl
# mkdir grpA
# mkdir grpB

# echo some_pid1 > grpA/tasks
# echo some_pid2 > grpA/tasks
# echo some_pid3 > grpA/tasks
# echo some_pid4 > grpA/tasks

...

# echo another_pidX > grpB/tasks
# echo another_pidY > grpB/tasks
```

All tasks in grpA/tasks should cumulatively share same CPU as all tasks
in grpB/tasks.

Signed-off-by : Srivatsa Vaddagiri <vatsa@linux.vnet.ibm.com>

```
 include/linux/container_subsys.h |    6
 init/Kconfig                     |   10 +
```

```
 kernel/sched.c                |  250 ++++++++++++++++++++++++++++++++++++++--
 kernel/sched_fair.c           |    3
 4 files changed, 256 insertions(+), 13 deletions(-)

Index: current/include/linux/container_subsys.h
===================================================================
--- current.orig/include/linux/container_subsys.h
+++ current/include/linux/container_subsys.h
@@ -24,3 +24,9 @@ SUBSYS(debug)
 #endif

 /* */
+
+#ifdef CONFIG_FAIR_GROUP_SCHED
+SUBSYS(cpuctlr)
+#endif
+
+/* */
Index: current/init/Kconfig
===================================================================
--- current.orig/init/Kconfig
+++ current/init/Kconfig
@@ -328,6 +328,16 @@ config CPUSETS

    Say N if unsure.

+config FAIR_GROUP_SCHED
+ bool "Fair group scheduler"
+ depends on EXPERIMENTAL
+ select CONTAINERS
+ help
+   This option enables you to group tasks and control CPU resource
+   allocation to such groups.
+
+   Say N if unsure.
+
 config SYSFS_DEPRECATED
  bool "Create deprecated sysfs files"
  default y
Index: current/kernel/sched.c
===================================================================
--- current.orig/kernel/sched.c
+++ current/kernel/sched.c
@@ -120,6 +120,56 @@ struct load_stat {
  u64 delta_fair, delta_exec, delta_stat;
 };

+#ifdef CONFIG_FAIR_GROUP_SCHED
```

```
+
+#include <linux/container.h>
+
+struct cfs_rq;
+
+/* task container/group related information */
+struct task_grp {
+ struct container_subsys_state css;
+ /* schedulable entities of this group on each cpu */
+ struct sched_entity **se;
+ /* runqueue "owned" by this group on each cpu */
+ struct cfs_rq **cfs_rq;
+};
+
+static DEFINE_PER_CPU(struct sched_entity, init_sched_entity);
+static DEFINE_PER_CPU(struct cfs_rq, init_cfs_rq) ____cacheline_aligned_in_smp;
+
+static struct sched_entity *init_sched_entity_p[CONFIG_NR_CPUS];
+static struct cfs_rq *init_cfs_rq_p[CONFIG_NR_CPUS];
+
+/* Default task group.
+ * Every task in system belong to this group at bootup and
+ * until administrator moves a task explicitly to another group.
+ */
+static struct task_grp init_task_grp =  {
+     .se     = init_sched_entity_p,
+     .cfs_rq = init_cfs_rq_p,
+     };
+
+/* return group to which a task belongs */
+static inline struct task_grp *task_grp(struct task_struct *p)
+{
+ return container_of(task_subsys_state(p, cpuctlr_subsys_id),
+    struct task_grp, css);
+}
+
+/* Change a task's cfs_rq and parent entity if it moves across CPUs/groups */
+static inline void set_task_cfs_rq(struct task_struct *p)
+{
+ p->se.cfs_rq = task_grp(p)->cfs_rq[task_cpu(p)];
+ p->se.parent = task_grp(p)->se[task_cpu(p)];
+}
+
+#else
+
+static inline void set_task_cfs_rq(struct task_struct *p) { }
+
+#endif /* CONFIG_FAIR_GROUP_SCHED */
```

```
+
 /* CFS-related fields in a runqueue */
 struct cfs_rq {
  struct load_weight load;
@@ -148,6 +198,7 @@ struct cfs_rq {
   * list is used during load balance.
   */
  struct list_head leaf_cfs_rq_list; /* Better name : task_cfs_rq_list? */
+ struct task_grp *tg;    /* group that "owns" this runqueue */
 #endif
 };

@@ -361,16 +412,6 @@ static inline unsigned long long rq_cloc
 #define task_rq(p)  cpu_rq(task_cpu(p))
 #define cpu_curr(cpu)  (cpu_rq(cpu)->curr)

-#ifdef CONFIG_FAIR_GROUP_SCHED
-/* Change a task's ->cfs_rq if it moves across CPUs */
-static inline void set_task_cfs_rq(struct task_struct *p)
-{
- p->se.cfs_rq = &task_rq(p)->cfs;
-}
-#else
-static inline void set_task_cfs_rq(struct task_struct *p) { }
-#endif
-
 #ifndef prepare_arch_switch
 # define prepare_arch_switch(next) do { } while (0)
 #endif
@@ -6232,7 +6273,23 @@ void __init sched_init(void)
   init_cfs_rq(&rq->cfs, rq);
 #ifdef CONFIG_FAIR_GROUP_SCHED
   INIT_LIST_HEAD(&rq->leaf_cfs_rq_list);
-  list_add(&rq->cfs.leaf_cfs_rq_list, &rq->leaf_cfs_rq_list);
+  {
+   struct cfs_rq *cfs_rq = &per_cpu(init_cfs_rq, i);
+   struct sched_entity *se =
+       &per_cpu(init_sched_entity, i);
+
+   init_cfs_rq_p[i] = cfs_rq;
+   init_cfs_rq(cfs_rq, rq);
+   cfs_rq->tg = &init_task_grp;
+   list_add(&cfs_rq->leaf_cfs_rq_list,
+        &rq->leaf_cfs_rq_list);
+
+   init_sched_entity_p[i] = se;
+   se->cfs_rq = &rq->cfs;
+   se->my_q = cfs_rq;
```

```
+    se->load.weight = NICE_0_LOAD;
+    se->parent = NULL;
+   }
 #endif

   for (j = 0; j < CPU_LOAD_IDX_MAX; j++)
@@ -6417,3 +6474,174 @@ void set_curr_task(int cpu, struct task_
 }

 #endif
+
+#ifdef CONFIG_FAIR_GROUP_SCHED
+
+/* return corresponding task_grp object of a container */
+static inline struct task_grp *container_tg(struct container *cont)
+{
+ return container_of(container_subsys_state(cont, cpuctlr_subsys_id),
+    struct task_grp, css);
+}
+
+/* allocate runqueue etc for a new task group */
+static int sched_create_group(struct container_subsys *ss,
+    struct container *cont)
+{
+ struct task_grp *tg;
+ struct cfs_rq *cfs_rq;
+ struct sched_entity *se;
+ int i;
+
+ if (!cont->parent) {
+  /* This is early initialization for the top container */
+  cont->subsys[cpuctlr_subsys_id] = &init_task_grp.css;
+  init_task_grp.css.container = cont;
+  return 0;
+ }
+
+ /* we support only 1-level deep hierarchical scheduler atm */
+ if (cont->parent->parent)
+  return -EINVAL;
+
+ tg = kzalloc(sizeof(*tg), GFP_KERNEL);
+ if (!tg)
+  return -ENOMEM;
+
+ tg->cfs_rq = kzalloc(sizeof(cfs_rq) * num_possible_cpus(), GFP_KERNEL);
+ if (!tg->cfs_rq)
+  goto err;
+ tg->se = kzalloc(sizeof(se) * num_possible_cpus(), GFP_KERNEL);
```

```
+ if (!tg->se)
+  goto err;
+
+ for_each_possible_cpu(i) {
+  struct rq *rq = cpu_rq(i);
+
+  cfs_rq = kmalloc_node(sizeof(struct cfs_rq), GFP_KERNEL,
+        cpu_to_node(i));
+  if (!cfs_rq)
+   goto err;
+
+  se = kmalloc_node(sizeof(struct sched_entity), GFP_KERNEL,
+       cpu_to_node(i));
+  if (!se)
+   goto err;
+
+  memset(cfs_rq, 0, sizeof(struct cfs_rq));
+  memset(se, 0, sizeof(struct sched_entity));
+
+  tg->cfs_rq[i] = cfs_rq;
+  init_cfs_rq(cfs_rq, rq);
+  cfs_rq->tg = tg;
+  list_add_rcu(&cfs_rq->leaf_cfs_rq_list, &rq->leaf_cfs_rq_list);
+
+  tg->se[i] = se;
+  se->cfs_rq = &rq->cfs;
+  se->my_q = cfs_rq;
+  se->load.weight = NICE_0_LOAD;
+  se->parent = NULL;
+ }
+
+ /* Bind the container to task_grp object we just created */
+ cont->subsys[cpuctlr_subsys_id] = &tg->css;
+ tg->css.container = cont;
+
+ return 0;
+
+err:
+ for_each_possible_cpu(i) {
+  if (tg->cfs_rq && tg->cfs_rq[i])
+   kfree(tg->cfs_rq[i]);
+  if (tg->se && tg->se[i])
+   kfree(tg->se[i]);
+ }
+ if (tg->cfs_rq)
+  kfree(tg->cfs_rq);
+ if (tg->se)
+  kfree(tg->se);
```

```
+ if (tg)
+  kfree(tg);
+
+ return -ENOMEM;
+}
+
+
+/* destroy runqueue etc associated with a task group */
+static void sched_destroy_group(struct container_subsys *ss,
+      struct container *cont)
+{
+ struct task_grp *tg = container_tg(cont);
+ struct cfs_rq *cfs_rq;
+ struct sched_entity *se;
+ int i;
+
+ for_each_possible_cpu(i) {
+  cfs_rq = tg->cfs_rq[i];
+  list_del_rcu(&cfs_rq->leaf_cfs_rq_list);
+ }
+
+ /* wait for possible concurrent references to cfs_rqs complete */
+ synchronize_sched();
+
+ /* now it should be safe to free those cfs_rqs */
+ for_each_possible_cpu(i) {
+  cfs_rq = tg->cfs_rq[i];
+  kfree(cfs_rq);
+
+  se = tg->se[i];
+  kfree(se);
+ }
+
+ kfree(tg);
+}
+
+/* change task's runqueue when it moves between groups */
+static void sched_move_task(struct container_subsys *ss, struct container *cont,
+    struct container *old_cont, struct task_struct *tsk)
+{
+ int on_rq;
+ unsigned long flags;
+ struct rq *rq;
+
+ rq = task_rq_lock(tsk, &flags);
+
+ on_rq = tsk->se.on_rq;
+ if (on_rq)
```

```
+  deactivate_task(rq, tsk, 0);
+
+ if (unlikely(rq->curr == tsk) && tsk->sched_class == &fair_sched_class)
+  tsk->sched_class->put_prev_task(rq, tsk, rq_clock(rq));
+
+ set_task_cfs_rq(tsk);
+
+ /* todo: change task's load_weight to reflect its new group */
+
+ if (on_rq)
+   activate_task(rq, tsk, 0);
+
+ if (unlikely(rq->curr == tsk) && tsk->sched_class == &fair_sched_class)
+  tsk->sched_class->set_curr_task(rq);
+
+ task_rq_unlock(rq, &flags);
+}
+
+
+static int sched_populate(struct container_subsys *ss, struct container *cont)
+{
+ /* todo: create a cpu_shares file to modify group weight */
+
+ return 0;
+}
+
+struct container_subsys cpuctlr_subsys = {
+ .name = "cpuctl",
+ .create = sched_create_group,
+ .destroy  = sched_destroy_group,
+ .attach = sched_move_task,
+ .populate = sched_populate,
+ .subsys_id = cpuctlr_subsys_id,
+ .early_init = 1,
+};
+
+#endif /* CONFIG_FAIR_GROUP_SCHED */
Index: current/kernel/sched_fair.c
===================================================================
--- current.orig/kernel/sched_fair.c
+++ current/kernel/sched_fair.c
@@ -743,8 +743,7 @@ static inline struct cfs_rq *group_cfs_r
  */
 static inline struct cfs_rq *cpu_cfs_rq(struct cfs_rq *cfs_rq, int this_cpu)
 {
- /* A later patch will take group into account */
- return &cpu_rq(this_cpu)->cfs;
+ return cfs_rq->tg->cfs_rq[this_cpu];
```

}

/* Iterate thr' all leaf cfs_rq's on a runqueue */

--
Regards,
vatsa

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers