Subject: [PATCH 1/5] Revert recent removal of set_curr_task()
Posted by Srivatsa Vaddagiri on Mon, 24 Sep 2007 16:28:26 GMT
View Forum Message <> Reply to Message

On Tue, Sep 18, 2007 at 09:36:30PM +0200, dimm wrote:
> here is a few cleanup/simplification/optimization(s)
> based on the recent modifications in the sched-dev tree.

[refer http://marc.info/?l=linux-kernel&m=119014449807290]

[snip]

> (3) rework enqueue/dequeue_entity() to get rid of
> sched_class::set_curr_task()

Dmitry/Ingo,
I am sorry for not having reviewed this change properly, but I
think we need to revert this.

Some problems with current sched-devel code introduced by this change:

1. SCHED_NORMAL->SCHED_RT transition of current task
a. we reset current->se.exec_start to 0 (update_stats_curr_end)
and not initialize it again. As a result, update_curr_rt()
can calculate bogus values

b. (for CONFIG_FAIR_GROUP_SCHED):
i) higher level entities of current task arent put back
on their cfs_rq
ii) their corresponding cfs_rq->curr is not set to NULL
iii) update_stats_wait_start() not called for higher
level entities of current task.

These functions are usually accomplished by put_prev_entity()

2. SCHED_RT->SCHED_NORMAL
a. (minor) we don't initialize se->prev_sum_exec_runtime
in enqueue_entity() under if (set_curr) { } block.
As a result the current task may get preempted
almost immediately?

b. (for CONFIG_FAIR_GROUP_SCHED):
i) higher level entities of current task arent taken out
of their cfs_rq
ii) their corresponding cfs_rq->curr is not set
iii) update_stats_wait_end() and update_stats_curr_start()
arent called for higher level entities of
current task.

These functions are usually accomplished by set_next_entity()

(similar problems exist while changing groups)

In theory its possible to solve these problems w/o reintroducing
set_curr_task(). I tried doing so, but found it clutters dequeue_entity
and enqueue_entity a lot and makes it less readable. It will duplicate what
put_prev_entity() and set_next_entity() are supposed to do. Moreoever
it is slightly inefficient to do all these in dequeue_entity() if we
consider that dequeue_entity can be called on current task for other
reasons as well (like when it is abt to sleep or change its nice value).

Considering these, IMHO, it's best to re-introduce set_curr_task()
and use put_prev_task/set_curr_task whenever the current task is
changing policies/groups.

Let me know what you think.

--

Revert removal of set_curr_task.
Use put_prev_task/set_curr_task when changing groups/policies

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

---
include/linux/sched.h | 1
kernel/sched.c | 34 ++++++++++++++++++------
kernel/sched_fair.c | 68 ++++++++++++++++++++++++++++++++-------------------
kernel/sched_idletask.c | 5 +++
kernel/sched_rt.c | 8 +++++
5 files changed, 72 insertions(+), 44 deletions(-)

Index: current/include/linux/sched.h
===
--- current.orig/include/linux/sched.h
+++ current/include/linux/sched.h
@@ -870,6 +870,7 @@ struct sched_class {
struct sched_domain *sd, enum cpu_idle_type idle,
int *all_pinned, int *this_best_prio);

+ void (*set_curr_task) (struct rq *rq);
void (*task_tick) (struct rq *rq, struct task_struct *p);
void (*task_new) (struct rq *rq, struct task_struct *p);

```
 };
Index: current/kernel/sched.c
===================================================================
--- current.orig/kernel/sched.c
+++ current/kernel/sched.c
@@ -3916,7 +3916,7 @@ EXPORT_SYMBOL(sleep_on_timeout);
 void rt_mutex_setprio(struct task_struct *p, int prio)
 {
  unsigned long flags;
- int oldprio, on_rq;
+ int oldprio, on_rq, running;
  struct rq *rq;

  BUG_ON(prio < 0 || prio > MAX_PRIO);
@@ -3926,8 +3926,12 @@ void rt_mutex_setprio(struct task_struct

  oldprio = p->prio;
  on_rq = p->se.on_rq;
- if (on_rq)
+ running = task_running(rq, p);
+ if (on_rq) {
   dequeue_task(rq, p, 0);
+  if (running)
+   p->sched_class->put_prev_task(rq, p);
+ }

  if (rt_prio(prio))
   p->sched_class = &rt_sched_class;
@@ -3937,13 +3941,15 @@ void rt_mutex_setprio(struct task_struct
  p->prio = prio;

  if (on_rq) {
+  if (running)
+   p->sched_class->set_curr_task(rq);
   enqueue_task(rq, p, 0);
   /*
    * Reschedule if we are currently running on this runqueue and
    * our priority decreased, or if we are not currently running on
    * this runqueue and our priority is higher than the current's
    */
-  if (task_running(rq, p)) {
+  if (running) {
    if (p->prio > oldprio)
     resched_task(rq->curr);
   } else {
@@ -4149,7 +4155,7 @@ __setscheduler(struct rq *rq, struct tas
 int sched_setscheduler(struct task_struct *p, int policy,
        struct sched_param *param)
```

```
 {
- int retval, oldprio, oldpolicy = -1, on_rq;
+ int retval, oldprio, oldpolicy = -1, on_rq, running;
  unsigned long flags;
  struct rq *rq;

@@ -4231,20 +4237,26 @@ recheck:
  }
  update_rq_clock(rq);
  on_rq = p->se.on_rq;
- if (on_rq)
+ running = task_running(rq, p);
+ if (on_rq) {
   deactivate_task(rq, p, 0);
+  if (running)
+   p->sched_class->put_prev_task(rq, p);
+ }

  oldprio = p->prio;
  __setscheduler(rq, p, policy, param->sched_priority);

  if (on_rq) {
+  if (running)
+   p->sched_class->set_curr_task(rq);
   activate_task(rq, p, 0);
   /*
    * Reschedule if we are currently running on this runqueue and
    * our priority decreased, or if we are not currently running on
    * this runqueue and our priority is higher than the current's
    */
-  if (task_running(rq, p)) {
+  if (running) {
    if (p->prio > oldprio)
     resched_task(rq->curr);
   } else {
@@ -6845,13 +6857,19 @@ static void sched_move_task(struct conta
  running = task_running(rq, tsk);
  on_rq = tsk->se.on_rq;

- if (on_rq)
+ if (on_rq) {
   dequeue_task(rq, tsk, 0);
+  if (unlikely(running))
+   tsk->sched_class->put_prev_task(rq, tsk);
+ }

  set_task_cfs_rq(tsk);
```

```
- if (on_rq)
+ if (on_rq) {
+  if (unlikely(running))
+   tsk->sched_class->set_curr_task(rq);
   enqueue_task(rq, tsk, 0);
+ }

 done:
  task_rq_unlock(rq, &flags);
Index: current/kernel/sched_fair.c
===================================================================
--- current.orig/kernel/sched_fair.c
+++ current/kernel/sched_fair.c
@@ -473,20 +473,9 @@ place_entity(struct cfs_rq *cfs_rq, stru
 }

 static void
-enqueue_entity(struct cfs_rq *cfs_rq, struct sched_entity *se,
-  int wakeup, int set_curr)
+enqueue_entity(struct cfs_rq *cfs_rq, struct sched_entity *se, int wakeup)
 {
  /*
-  * In case of the 'current'.
-  */
- if (unlikely(set_curr)) {
-  update_stats_curr_start(cfs_rq, se);
-  cfs_rq->curr = se;
-  account_entity_enqueue(cfs_rq, se);
-  return;
- }
-
- /*
   * Update the fair clock.
   */
  update_curr(cfs_rq);
@@ -497,7 +486,8 @@ enqueue_entity(struct cfs_rq *cfs_rq, st
  }

  update_stats_enqueue(cfs_rq, se);
- __enqueue_entity(cfs_rq, se);
+ if (se != cfs_rq->curr)
+  __enqueue_entity(cfs_rq, se);
  account_entity_enqueue(cfs_rq, se);
 }

@@ -517,12 +507,8 @@ dequeue_entity(struct cfs_rq *cfs_rq, st
  }
 }
```

```
 #endif
- if (likely(se != cfs_rq->curr))
+ if (se != cfs_rq->curr)
    __dequeue_entity(cfs_rq, se);
- else {
-  update_stats_curr_end(cfs_rq, se);
-  cfs_rq->curr = NULL;
- }
  account_entity_dequeue(cfs_rq, se);
 }

@@ -540,15 +526,20 @@ check_preempt_tick(struct cfs_rq *cfs_rq
   resched_task(rq_of(cfs_rq)->curr);
 }

-static inline void
+static void
 set_next_entity(struct cfs_rq *cfs_rq, struct sched_entity *se)
 {
- /*
-  * Any task has to be enqueued before it get to execute on
-  * a CPU. So account for the time it spent waiting on the
-  * runqueue.
-  */
- update_stats_wait_end(cfs_rq, se);
+ /* 'current' is not kept within the tree. */
+ if (se->on_rq) {
+  /*
+   * Any task has to be enqueued before it get to execute on
+   * a CPU. So account for the time it spent waiting on the
+   * runqueue.
+   */
+  update_stats_wait_end(cfs_rq, se);
+  __dequeue_entity(cfs_rq, se);
+ }
+
  update_stats_curr_start(cfs_rq, se);
  cfs_rq->curr = se;
 #ifdef CONFIG_SCHEDSTATS
@@ -569,10 +560,6 @@ static struct sched_entity *pick_next_en
 {
  struct sched_entity *se = __pick_next_entity(cfs_rq);

- /* 'current' is not kept within the tree. */
- if (se)
-  __dequeue_entity(cfs_rq, se);
-
  set_next_entity(cfs_rq, se);
```

```
  return se;
@@ -704,17 +691,12 @@ static void enqueue_task_fair(struct rq
 {
  struct cfs_rq *cfs_rq;
  struct sched_entity *se = &p->se;
- int set_curr = 0;
-
- /* Are we enqueuing the current task? */
- if (unlikely(task_running(rq, p)))
-  set_curr = 1;

  for_each_sched_entity(se) {
   if (se->on_rq)
    break;
   cfs_rq = cfs_rq_of(se);
-  enqueue_entity(cfs_rq, se, wakeup, set_curr);
+  enqueue_entity(cfs_rq, se, wakeup);
  }
 }

@@ -762,7 +744,7 @@ static void yield_task_fair(struct rq *r
    * position within the tree:
    */
   dequeue_entity(cfs_rq, se, 0);
-  enqueue_entity(cfs_rq, se, 0, 1);
+  enqueue_entity(cfs_rq, se, 0);

   return;
  }
@@ -1005,6 +987,19 @@ static void task_new_fair(struct rq *rq,
  resched_task(rq->curr);
 }

+/* Account for a task changing its policy or group.
+ *
+ * This routine is mostly called to set cfs_rq->curr field when a task
+ * migrates between groups/classes.
+ */
+static void set_curr_task_fair(struct rq *rq)
+{
+ struct sched_entity *se = &rq->curr->se;
+
+ for_each_sched_entity(se)
+  set_next_entity(cfs_rq_of(se), se);
+}
+
 /*
```

```
  * All the scheduling class methods:
  */
@@ -1020,6 +1015,7 @@ struct sched_class fair_sched_class __re

  .load_balance  = load_balance_fair,

+ .set_curr_task          = set_curr_task_fair,
  .task_tick  = task_tick_fair,
  .task_new  = task_new_fair,
 };
Index: current/kernel/sched_idletask.c
===================================================================
--- current.orig/kernel/sched_idletask.c
+++ current/kernel/sched_idletask.c
@@ -50,6 +50,10 @@ static void task_tick_idle(struct rq *rq
 {
 }

+static void set_curr_task_idle(struct rq *rq)
+{
+}
+
 /*
  * Simple, special scheduling class for the per-CPU idle tasks:
  */
@@ -66,6 +70,7 @@ static struct sched_class idle_sched_cla

  .load_balance  = load_balance_idle,

+ .set_curr_task          = set_curr_task_idle,
  .task_tick  = task_tick_idle,
  /* no .task_new for idle tasks */
 };
Index: current/kernel/sched_rt.c
===================================================================
--- current.orig/kernel/sched_rt.c
+++ current/kernel/sched_rt.c
@@ -218,6 +218,13 @@ static void task_tick_rt(struct rq *rq,
  }
 }

+static void set_curr_task_rt(struct rq *rq)
+{
+ struct task_struct *p = rq->curr;
+
+ p->se.exec_start = rq->clock;
+}
+
```

```
 static struct sched_class rt_sched_class __read_mostly = {
  .enqueue_task  = enqueue_task_rt,
  .dequeue_task  = dequeue_task_rt,
@@ -230,5 +237,6 @@ static struct sched_class rt_sched_class

  .load_balance  = load_balance_rt,

+ .set_curr_task          = set_curr_task_rt,
  .task_tick  = task_tick_rt,
 };
```

--
Regards,
vatsa