Subject: [PATCH v4 3/4] mark whenever put_prev_task is called from context_switch
Posted by Glauber Costa on Tue, 05 Jun 2012 14:49:40 GMT
View Forum Message <> Reply to Message

In the interest of having the number of context switches stored per-group,
this patch introduces a boolean argument to put_prev_task. It is needed
because put_prev_task is called from many sites, not all of them denoting
a context switch.

This would be needed even if pick_next_task has the behavior of put_prev_task
embedded into it: This is because we may be picking a task from a different
class, and that would translate to a raw call to put_prev_task.

For the fair class, nr_switches is stored for all the hierarchy. This is
because a walk is done anyway there, so we just reuse it.

For the rt class, we only store in the current group, therefore not
forcing a walk. Readers then may have to accumulate.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
CC: Paul Turner <pjt@google.com>
---
 include/linux/sched.h | 2 +-
 kernel/sched/core.c | 11 ++++++++-----
 kernel/sched/fair.c | 4 +++-
 kernel/sched/idle_task.c | 2 +-
 kernel/sched/rt.c | 6 +++++-
 kernel/sched/sched.h | 3 +++
 kernel/sched/stop_task.c | 2 +-
 7 files changed, 20 insertions(+), 10 deletions(-)

diff --git a/include/linux/sched.h b/include/linux/sched.h
index f45c0b2..2bff097 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1083,7 +1083,7 @@ struct sched_class {
  void (*check_preempt_curr) (struct rq *rq, struct task_struct *p, int flags);

  struct task_struct * (*pick_next_task) (struct rq *rq);
- void (*put_prev_task) (struct rq *rq, struct task_struct *p);
+ void (*put_prev_task) (struct rq *rq, struct task_struct *p, bool ctxsw);

 #ifdef CONFIG_SMP
  int (*select_task_rq)(struct task_struct *p, int sd_flag, int flags);
diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index c058189..6803fd1 100644

```
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -3164,7 +3164,7 @@ static void put_prev_task(struct rq *rq, struct task_struct *prev)
 {
  if (prev->on_rq || rq->skip_clock_update < 0)
   update_rq_clock(rq);
- prev->sched_class->put_prev_task(rq, prev);
+ prev->sched_class->put_prev_task(rq, prev, true);
 }

 /*
@@ -3248,6 +3248,7 @@ need_resched:
  if (unlikely(!rq->nr_running))
   idle_balance(cpu, rq);

+
  put_prev_task(rq, prev);
  next = pick_next_task(rq);
  clear_tsk_need_resched(prev);
@@ -3857,7 +3858,7 @@ void rt_mutex_setprio(struct task_struct *p, int prio)
  if (on_rq)
   dequeue_task(rq, p, 0);
  if (running)
-  p->sched_class->put_prev_task(rq, p);
+  p->sched_class->put_prev_task(rq, p, false);

  if (rt_prio(prio))
   p->sched_class = &rt_sched_class;
@@ -4208,7 +4209,7 @@ recheck:
  if (on_rq)
   dequeue_task(rq, p, 0);
  if (running)
-  p->sched_class->put_prev_task(rq, p);
+  p->sched_class->put_prev_task(rq, p, false);

  p->sched_reset_on_fork = reset_on_fork;

@@ -5197,7 +5198,7 @@ static void migrate_tasks(unsigned int dead_cpu)

   next = pick_next_task(rq);
   BUG_ON(!next);
-  next->sched_class->put_prev_task(rq, next);
+  next->sched_class->put_prev_task(rq, next, false);

   /* Find suitable destination for @next, with force if needed. */
   dest_cpu = select_fallback_rq(dead_cpu, next);
@@ -7318,7 +7319,7 @@ void sched_move_task(struct task_struct *tsk)
  if (on_rq)
```

```
   dequeue_task(rq, tsk, 0);
  if (unlikely(running))
-  tsk->sched_class->put_prev_task(rq, tsk);
+  tsk->sched_class->put_prev_task(rq, tsk, false);

 #ifdef CONFIG_FAIR_GROUP_SCHED
  if (tsk->sched_class->task_move_group)
diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c
index 940e6d1..1c8a04e 100644
--- a/kernel/sched/fair.c
+++ b/kernel/sched/fair.c
@@ -2996,7 +2996,7 @@ static struct task_struct *pick_next_task_fair(struct rq *rq)
 /*
  * Account for a descheduled task:
  */
-static void put_prev_task_fair(struct rq *rq, struct task_struct *prev)
+static void put_prev_task_fair(struct rq *rq, struct task_struct *prev, bool ctxsw)
 {
  struct sched_entity *se = &prev->se;
  struct cfs_rq *cfs_rq;
@@ -3004,6 +3004,8 @@ static void put_prev_task_fair(struct rq *rq, struct task_struct *prev)
  for_each_sched_entity(se) {
   cfs_rq = cfs_rq_of(se);
   put_prev_entity(cfs_rq, se);
+  if (ctxsw)
+   cfs_rq->nr_switches++;
  }
 }

diff --git a/kernel/sched/idle_task.c b/kernel/sched/idle_task.c
index b44d604..08c3ad3 100644
--- a/kernel/sched/idle_task.c
+++ b/kernel/sched/idle_task.c
@@ -42,7 +42,7 @@ dequeue_task_idle(struct rq *rq, struct task_struct *p, int flags)
  raw_spin_lock_irq(&rq->lock);
 }

-static void put_prev_task_idle(struct rq *rq, struct task_struct *prev)
+static void put_prev_task_idle(struct rq *rq, struct task_struct *prev, bool ctxsw)
 {
 }

diff --git a/kernel/sched/rt.c b/kernel/sched/rt.c
index c5565c3..afe5096 100644
--- a/kernel/sched/rt.c
+++ b/kernel/sched/rt.c
@@ -1387,8 +1387,10 @@ static struct task_struct *pick_next_task_rt(struct rq *rq)
  return p;
```

```
 }

-static void put_prev_task_rt(struct rq *rq, struct task_struct *p)
+static void put_prev_task_rt(struct rq *rq, struct task_struct *p, bool ctxsw)
 {
+ struct sched_rt_entity *rt_se = &p->rt;
+ struct rt_rq *rt_rq = rt_rq_of_se(rt_se);
  update_curr_rt(rq);

  /*
@@ -1397,6 +1399,8 @@ static void put_prev_task_rt(struct rq *rq, struct task_struct *p)
   */
  if (on_rt_rq(&p->rt) && p->rt.nr_cpus_allowed > 1)
   enqueue_pushable_task(rq, p);
+ if (ctxsw)
+  rt_rq->rt_nr_switches++;
 }

 #ifdef CONFIG_SMP
diff --git a/kernel/sched/sched.h b/kernel/sched/sched.h
index ba9dccf..7548814 100644
--- a/kernel/sched/sched.h
+++ b/kernel/sched/sched.h
@@ -237,6 +237,7 @@ struct cfs_rq {
  struct list_head leaf_cfs_rq_list;
  struct task_group *tg; /* group that "owns" this runqueue */

+ u64 nr_switches;
 #ifdef CONFIG_SMP
  /*
   *   h_load = weight * f(tg)
@@ -306,6 +307,8 @@ struct rt_rq {
  struct rq *rq;
  struct list_head leaf_rt_rq_list;
  struct task_group *tg;
+
+ u64 rt_nr_switches;
 #endif
 };

diff --git a/kernel/sched/stop_task.c b/kernel/sched/stop_task.c
index 7b386e8..19ff22d 100644
--- a/kernel/sched/stop_task.c
+++ b/kernel/sched/stop_task.c
@@ -50,7 +50,7 @@ static void yield_task_stop(struct rq *rq)
  BUG(); /* the stop task should never yield, its pointless. */
 }
```

```
-static void put_prev_task_stop(struct rq *rq, struct task_struct *prev)
+static void put_prev_task_stop(struct rq *rq, struct task_struct *prev, bool ctxsw)
 {
 }

--
1.7.10.2
```