Subject: [RFC][PATCH 1/6] Introduce struct sched_entity and struct lrq
Posted by Srivatsa Vaddagiri on Mon, 11 Jun 2007 15:50:46 GMT
View Forum Message <> Reply to Message

This patch introduces two new structures:

struct sched_entity
	stores essential attributes/execution-history used by CFS core
	to drive fairness between 'schedulable entities' (tasks, users etc)

struct lrq
	runqueue used to hold ready-to-run entities

These new structures are formed by grouping together existing fields in
existing structures (task_struct and rq) and hence represents rework
with zero functionality change.

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

```
---
 fs/proc/array.c           |    4
 include/linux/sched.h     |   43 +++++----
 kernel/exit.c             |    2
 kernel/posix-cpu-timers.c |   16 +--
 kernel/sched.c            |  186 +++++++++++++++++++-------------
 kernel/sched_debug.c      |   86 +++++++++---------
 kernel/sched_fair.c       |  211 +++++++++++++++++++++++----------------------
 kernel/sched_rt.c         |   14 +--
 8 files changed, 289 insertions(+), 273 deletions(-)

Index: current/include/linux/sched.h
===================================================================
--- current.orig/include/linux/sched.h 2007-06-09 15:01:39.000000000 +0530
+++ current/include/linux/sched.h 2007-06-09 15:04:54.000000000 +0530
@@ -872,6 +872,29 @@
  void (*task_new) (struct rq *rq, struct task_struct *p);
 };

+/* CFS stats for a schedulable entity (task, task-group etc) */
+struct sched_entity {
+ int load_weight; /* for niceness load balancing purposes */
+ int on_rq;
+ struct rb_node run_node;
+ u64 wait_start_fair;
+ u64 wait_start;
+ u64 exec_start;
+ u64 sleep_start, sleep_start_fair;
```

```
+ u64 block_start;
+ u64 sleep_max;
+ u64 block_max;
+ u64 exec_max;
+ u64 wait_max;
+ u64 last_ran;
+
+ s64 wait_runtime;
+ u64 sum_exec_runtime;
+ s64 fair_key;
+ s64 sum_wait_runtime, sum_sleep_runtime;
+ unsigned long wait_runtime_overruns, wait_runtime_underruns;
+};
+
 struct task_struct {
  volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
  void *stack;
@@ -886,33 +909,15 @@
  int oncpu;
 #endif
 #endif
- int load_weight; /* for niceness load balancing purposes */

  int prio, static_prio, normal_prio;
- int on_rq;
  struct list_head run_list;
- struct rb_node run_node;
+ struct sched_entity se;

  unsigned short ioprio;
 #ifdef CONFIG_BLK_DEV_IO_TRACE
  unsigned int btrace_seq;
 #endif
- /* CFS scheduling class statistics fields: */
- u64 wait_start_fair;
- u64 wait_start;
- u64 exec_start;
- u64 sleep_start, sleep_start_fair;
- u64 block_start;
- u64 sleep_max;
- u64 block_max;
- u64 exec_max;
- u64 wait_max;
-
- s64 wait_runtime;
- u64 sum_exec_runtime;
- s64 fair_key;
- s64 sum_wait_runtime, sum_sleep_runtime;
```

```
- unsigned long wait_runtime_overruns, wait_runtime_underruns;

  unsigned int policy;
  cpumask_t cpus_allowed;
Index: current/fs/proc/array.c
===================================================================
--- current.orig/fs/proc/array.c 2007-06-09 15:01:39.000000000 +0530
+++ current/fs/proc/array.c 2007-06-09 15:04:54.000000000 +0530
@@ -329,7 +329,7 @@
   * Use CFS's precise accounting, if available:
   */
  if (!(sysctl_sched_features & 128)) {
-  u64 temp = (u64)nsec_to_clock_t(p->sum_exec_runtime);
+  u64 temp = (u64)nsec_to_clock_t(p->se.sum_exec_runtime);

   if (total) {
    temp *= utime;
@@ -351,7 +351,7 @@
   * by userspace grows monotonically - apps rely on that):
   */
  if (!(sysctl_sched_features & 128))
-  stime = nsec_to_clock_t(p->sum_exec_runtime) - task_utime(p);
+  stime = nsec_to_clock_t(p->se.sum_exec_runtime) - task_utime(p);

  return stime;
 }
Index: current/kernel/exit.c
===================================================================
--- current.orig/kernel/exit.c 2007-06-09 14:56:50.000000000 +0530
+++ current/kernel/exit.c 2007-06-09 15:04:54.000000000 +0530
@@ -126,7 +126,7 @@
   sig->nivcsw += tsk->nivcsw;
   sig->inblock += task_io_get_inblock(tsk);
   sig->oublock += task_io_get_oublock(tsk);
-  sig->sum_sched_runtime += tsk->sum_exec_runtime;
+  sig->sum_sched_runtime += tsk->se.sum_exec_runtime;
   sig = NULL; /* Marker for below. */
  }

Index: current/kernel/posix-cpu-timers.c
===================================================================
--- current.orig/kernel/posix-cpu-timers.c 2007-06-09 15:01:39.000000000 +0530
+++ current/kernel/posix-cpu-timers.c 2007-06-09 15:04:54.000000000 +0530
@@ -249,7 +249,7 @@
   cpu->sched = p->signal->sum_sched_runtime;
   /* Add in each other live thread.  */
   while ((t = next_thread(t)) != p) {
-   cpu->sched += t->sum_exec_runtime;
```

```
+   cpu->sched += t->se.sum_exec_runtime;
   }
   cpu->sched += sched_ns(p);
   break;
@@ -467,7 +467,7 @@
 void posix_cpu_timers_exit(struct task_struct *tsk)
 {
  cleanup_timers(tsk->cpu_timers,
-        tsk->utime, tsk->stime, tsk->sum_exec_runtime);
+        tsk->utime, tsk->stime, tsk->se.sum_exec_runtime);

 }
 void posix_cpu_timers_exit_group(struct task_struct *tsk)
@@ -475,7 +475,7 @@
  cleanup_timers(tsk->signal->cpu_timers,
         cputime_add(tsk->utime, tsk->signal->utime),
         cputime_add(tsk->stime, tsk->signal->stime),
-        tsk->sum_exec_runtime + tsk->signal->sum_sched_runtime);
+       tsk->se.sum_exec_runtime + tsk->signal->sum_sched_runtime);
 }


@@ -536,7 +536,7 @@
   nsleft = max_t(unsigned long long, nsleft, 1);
   do {
    if (likely(!(t->flags & PF_EXITING))) {
-    ns = t->sum_exec_runtime + nsleft;
+    ns = t->se.sum_exec_runtime + nsleft;
     if (t->it_sched_expires == 0 ||
         t->it_sched_expires > ns) {
      t->it_sched_expires = ns;
@@ -1004,7 +1004,7 @@
   struct cpu_timer_list *t = list_first_entry(timers,
             struct cpu_timer_list,
             entry);
-  if (!--maxfire || tsk->sum_exec_runtime < t->expires.sched) {
+  if (!--maxfire || tsk->se.sum_exec_runtime < t->expires.sched) {
    tsk->it_sched_expires = t->expires.sched;
    break;
   }
@@ -1049,7 +1049,7 @@
  do {
   utime = cputime_add(utime, t->utime);
   stime = cputime_add(stime, t->stime);
-  sum_sched_runtime += t->sum_exec_runtime;
+  sum_sched_runtime += t->se.sum_exec_runtime;
   t = next_thread(t);
  } while (t != tsk);
```

```
  ptime = cputime_add(utime, stime);
@@ -1208,7 +1208,7 @@
     t->it_virt_expires = ticks;
    }

-   sched = t->sum_exec_runtime + sched_left;
+   sched = t->se.sum_exec_runtime + sched_left;
    if (sched_expires && (t->it_sched_expires == 0 ||
            t->it_sched_expires > sched)) {
     t->it_sched_expires = sched;
@@ -1300,7 +1300,7 @@

  if (UNEXPIRED(prof) && UNEXPIRED(virt) &&
      (tsk->it_sched_expires == 0 ||
-      tsk->sum_exec_runtime < tsk->it_sched_expires))
+      tsk->se.sum_exec_runtime < tsk->it_sched_expires))
   return;

 #undef UNEXPIRED
Index: current/kernel/sched.c
===================================================================
--- current.orig/kernel/sched.c 2007-06-09 15:01:39.000000000 +0530
+++ current/kernel/sched.c 2007-06-09 15:07:17.000000000 +0530
@@ -116,6 +116,23 @@
  struct list_head queue[MAX_RT_PRIO];
 };

+/* CFS-related fields in a runqueue */
+struct lrq {
+ unsigned long raw_weighted_load;
+ #define CPU_LOAD_IDX_MAX 5
+ unsigned long cpu_load[CPU_LOAD_IDX_MAX];
+ unsigned long nr_load_updates;
+
+ u64 fair_clock, delta_fair_clock;
+ u64 exec_clock, delta_exec_clock;
+ s64 wait_runtime;
+ unsigned long wait_runtime_overruns, wait_runtime_underruns;
+
+ struct rb_root tasks_timeline;
+ struct rb_node *rb_leftmost;
+ struct rb_node *rb_load_balance_curr;
+};
+
 /*
  * This is the main, per-CPU runqueue data structure.
  *
@@ -131,16 +148,13 @@
```

```
   * remote CPUs use both these fields when doing load calculation.
   */
  long nr_running;
- unsigned long raw_weighted_load;
- #define CPU_LOAD_IDX_MAX 5
- unsigned long cpu_load[CPU_LOAD_IDX_MAX];
+ struct lrq lrq;

  unsigned char idle_at_tick;
 #ifdef CONFIG_NO_HZ
  unsigned char in_nohz_recently;
 #endif
  u64 nr_switches;
- unsigned long nr_load_updates;

  /*
   * This is part of a global counter where only the total sum
@@ -156,10 +170,6 @@

  u64 clock, prev_clock_raw;
  s64 clock_max_delta;
- u64 fair_clock, delta_fair_clock;
- u64 exec_clock, delta_exec_clock;
- s64 wait_runtime;
- unsigned long wait_runtime_overruns, wait_runtime_underruns;

  unsigned int clock_warps, clock_overflows;
  unsigned int clock_unstable_events;
@@ -170,10 +180,6 @@
  int rt_load_balance_idx;
  struct list_head *rt_load_balance_head, *rt_load_balance_curr;

- struct rb_root tasks_timeline;
- struct rb_node *rb_leftmost;
- struct rb_node *rb_load_balance_curr;
-
  atomic_t nr_iowait;

 #ifdef CONFIG_SMP
@@ -583,13 +589,13 @@
 static inline void
 inc_raw_weighted_load(struct rq *rq, const struct task_struct *p)
 {
- rq->raw_weighted_load += p->load_weight;
+ rq->lrq.raw_weighted_load += p->se.load_weight;
 }

 static inline void
```

```
 dec_raw_weighted_load(struct rq *rq, const struct task_struct *p)
 {
- rq->raw_weighted_load -= p->load_weight;
+ rq->lrq.raw_weighted_load -= p->se.load_weight;
 }

 static inline void inc_nr_running(struct task_struct *p, struct rq *rq)
@@ -615,22 +621,22 @@

 static void set_load_weight(struct task_struct *p)
 {
- task_rq(p)->wait_runtime -= p->wait_runtime;
- p->wait_runtime = 0;
+ task_rq(p)->lrq.wait_runtime -= p->se.wait_runtime;
+ p->se.wait_runtime = 0;

  if (has_rt_policy(p)) {
-  p->load_weight = prio_to_weight[0] * 2;
+  p->se.load_weight = prio_to_weight[0] * 2;
   return;
  }
  /*
   * SCHED_IDLEPRIO tasks get minimal weight:
   */
  if (p->policy == SCHED_IDLEPRIO) {
-  p->load_weight = 1;
+  p->se.load_weight = 1;
   return;
  }

- p->load_weight = prio_to_weight[p->static_prio - MAX_RT_PRIO];
+ p->se.load_weight = prio_to_weight[p->static_prio - MAX_RT_PRIO];
 }

 static void enqueue_task(struct rq *rq, struct task_struct *p, int wakeup)
@@ -639,7 +645,7 @@

  sched_info_queued(p);
  p->sched_class->enqueue_task(rq, p, wakeup, now);
- p->on_rq = 1;
+ p->se.on_rq = 1;
 }

 static void dequeue_task(struct rq *rq, struct task_struct *p, int sleep)
@@ -647,7 +653,7 @@
  u64 now = rq_clock(rq);

  p->sched_class->dequeue_task(rq, p, sleep, now);
```

```
- p->on_rq = 0;
+ p->se.on_rq = 0;
 }

 /*
@@ -735,7 +741,7 @@
 /* Used instead of source_load when we know the type == 0 */
 unsigned long weighted_cpuload(const int cpu)
 {
- return cpu_rq(cpu)->raw_weighted_load;
+ return cpu_rq(cpu)->lrq.raw_weighted_load;
 }

 #ifdef CONFIG_SMP
@@ -752,18 +758,18 @@
  u64 clock_offset, fair_clock_offset;

  clock_offset = old_rq->clock - new_rq->clock;
- fair_clock_offset = old_rq->fair_clock - new_rq->fair_clock;
+ fair_clock_offset = old_rq->lrq.fair_clock - new_rq->lrq.fair_clock;

- if (p->wait_start)
-  p->wait_start -= clock_offset;
- if (p->wait_start_fair)
-  p->wait_start_fair -= fair_clock_offset;
- if (p->sleep_start)
-  p->sleep_start -= clock_offset;
- if (p->block_start)
-  p->block_start -= clock_offset;
- if (p->sleep_start_fair)
-  p->sleep_start_fair -= fair_clock_offset;
+ if (p->se.wait_start)
+  p->se.wait_start -= clock_offset;
+ if (p->se.wait_start_fair)
+  p->se.wait_start_fair -= fair_clock_offset;
+ if (p->se.sleep_start)
+  p->se.sleep_start -= clock_offset;
+ if (p->se.block_start)
+  p->se.block_start -= clock_offset;
+ if (p->se.sleep_start_fair)
+  p->se.sleep_start_fair -= fair_clock_offset;

  task_thread_info(p)->cpu = new_cpu;

@@ -791,7 +797,7 @@
   * If the task is not on a runqueue (and not running), then
   * it is sufficient to simply update the task's cpu field.
   */
```

```
- if (!p->on_rq && !task_running(rq, p)) {
+ if (!p->se.on_rq && !task_running(rq, p)) {
   set_task_cpu(p, dest_cpu);
   return 0;
  }
@@ -822,7 +828,7 @@
 repeat:
  rq = task_rq_lock(p, &flags);
  /* Must be off runqueue entirely, not preempted. */
- if (unlikely(p->on_rq || task_running(rq, p))) {
+ if (unlikely(p->se.on_rq || task_running(rq, p))) {
   /* If it's preempted, we yield.  It could be a while. */
   preempted = !task_running(rq, p);
   task_rq_unlock(rq, &flags);
@@ -870,9 +876,9 @@
  struct rq *rq = cpu_rq(cpu);

  if (type == 0)
-  return rq->raw_weighted_load;
+  return rq->lrq.raw_weighted_load;

- return min(rq->cpu_load[type-1], rq->raw_weighted_load);
+ return min(rq->lrq.cpu_load[type-1], rq->lrq.raw_weighted_load);
 }

 /*
@@ -884,9 +890,9 @@
  struct rq *rq = cpu_rq(cpu);

  if (type == 0)
-  return rq->raw_weighted_load;
+  return rq->lrq.raw_weighted_load;

- return max(rq->cpu_load[type-1], rq->raw_weighted_load);
+ return max(rq->lrq.cpu_load[type-1], rq->lrq.raw_weighted_load);
 }

 /*
@@ -897,7 +903,7 @@
  struct rq *rq = cpu_rq(cpu);
  unsigned long n = rq->nr_running;

- return n ? rq->raw_weighted_load / n : SCHED_LOAD_SCALE;
+ return n ? rq->lrq.raw_weighted_load / n : SCHED_LOAD_SCALE;
 }

 /*
@@ -1128,7 +1134,7 @@
```

```
  if (!(old_state & state))
   goto out;

- if (p->on_rq)
+ if (p->se.on_rq)
   goto out_running;

  cpu = task_cpu(p);
@@ -1183,11 +1189,11 @@
     * of the current CPU:
     */
    if (sync)
-    tl -= current->load_weight;
+    tl -= current->se.load_weight;

    if ((tl <= load &&
     tl + target_load(cpu, idx) <= tl_per_task) ||
-    100*(tl + p->load_weight) <= imbalance*load) {
+          100*(tl + p->se.load_weight) <= imbalance*load) {
     /*
      * This domain has SD_WAKE_AFFINE and
      * p is cache cold in this domain, and
@@ -1221,7 +1227,7 @@
   old_state = p->state;
   if (!(old_state & state))
    goto out;
-  if (p->on_rq)
+  if (p->se.on_rq)
    goto out_running;

   this_cpu = smp_processor_id();
@@ -1285,18 +1291,18 @@
  */
 static void __sched_fork(struct task_struct *p)
 {
- p->wait_start_fair = p->wait_start = p->exec_start = 0;
- p->sum_exec_runtime = 0;
+ p->se.wait_start_fair = p->se.wait_start = p->se.exec_start = 0;
+ p->se.sum_exec_runtime = 0;

- p->wait_runtime = 0;
+ p->se.wait_runtime = 0;

- p->sum_wait_runtime = p->sum_sleep_runtime = 0;
- p->sleep_start = p->sleep_start_fair = p->block_start = 0;
- p->sleep_max = p->block_max = p->exec_max = p->wait_max = 0;
- p->wait_runtime_overruns = p->wait_runtime_underruns = 0;
+ p->se.sum_wait_runtime = p->se.sum_sleep_runtime = 0;
```

```
+ p->se.sleep_start = p->se.sleep_start_fair = p->se.block_start = 0;
+ p->se.sleep_max = p->se.block_max = p->se.exec_max = p->se.wait_max = 0;
+ p->se.wait_runtime_overruns = p->se.wait_runtime_underruns = 0;

  INIT_LIST_HEAD(&p->run_list);
- p->on_rq = 0;
+ p->se.on_rq = 0;
  p->nr_switches = 0;

  /*
@@ -1367,7 +1373,7 @@
  p->prio = effective_prio(p);

  if (!sysctl_sched_child_runs_first || (clone_flags & CLONE_VM) ||
-   task_cpu(p) != this_cpu || !current->on_rq) {
+   task_cpu(p) != this_cpu || !current->se.on_rq) {
   activate_task(rq, p, 0);
  } else {
   /*
@@ -1382,7 +1388,7 @@

 void sched_dead(struct task_struct *p)
 {
- WARN_ON_ONCE(p->on_rq);
+ WARN_ON_ONCE(p->se.on_rq);
 }

 /**
@@ -1592,17 +1598,17 @@
  u64 fair_delta64, exec_delta64, tmp64;
  unsigned int i, scale;

- this_rq->nr_load_updates++;
+ this_rq->lrq.nr_load_updates++;
  if (!(sysctl_sched_features & 64)) {
-  this_load = this_rq->raw_weighted_load;
+  this_load = this_rq->lrq.raw_weighted_load;
   goto do_avg;
  }

- fair_delta64 = this_rq->delta_fair_clock + 1;
- this_rq->delta_fair_clock = 0;
+ fair_delta64 = this_rq->lrq.delta_fair_clock + 1;
+ this_rq->lrq.delta_fair_clock = 0;

- exec_delta64 = this_rq->delta_exec_clock + 1;
- this_rq->delta_exec_clock = 0;
+ exec_delta64 = this_rq->lrq.delta_exec_clock + 1;
```

```
+ this_rq->lrq.delta_exec_clock = 0;

  if (fair_delta64 > (u64)LONG_MAX)
   fair_delta64 = (u64)LONG_MAX;
@@ -1627,10 +1633,10 @@

   /* scale is effectively 1 << i now, and >> i divides by scale */

-  old_load = this_rq->cpu_load[i];
+  old_load = this_rq->lrq.cpu_load[i];
   new_load = this_load;

-  this_rq->cpu_load[i] = (old_load*(scale-1) + new_load) >> i;
+  this_rq->lrq.cpu_load[i] = (old_load*(scale-1) + new_load) >> i;
  }
 }

@@ -1886,7 +1892,8 @@
   * skip a task if it will be the highest priority task (i.e. smallest
   * prio value) on its new queue regardless of its load weight
   */
- skip_for_load = (p->load_weight >> 1) > rem_load_move + SCHED_LOAD_SCALE_FUZZ;
+ skip_for_load = (p->se.load_weight >> 1) > rem_load_move +
+         SCHED_LOAD_SCALE_FUZZ;
  if (skip_for_load && p->prio < this_best_prio)
   skip_for_load = !best_prio_seen && p->prio == best_prio;
  if (skip_for_load ||
@@ -1899,7 +1906,7 @@

  pull_task(busiest, p, this_rq, this_cpu);
  pulled++;
- rem_load_move -= p->load_weight;
+ rem_load_move -= p->se.load_weight;

  /*
   * We only want to steal up to the prescribed number of tasks
@@ -1996,7 +2003,7 @@

    avg_load += load;
    sum_nr_running += rq->nr_running;
-   sum_weighted_load += rq->raw_weighted_load;
+   sum_weighted_load += rq->lrq.raw_weighted_load;
   }

   /*
@@ -2230,11 +2237,12 @@

   rq = cpu_rq(i);
```

```
-  if (rq->nr_running == 1 && rq->raw_weighted_load > imbalance)
+  if (rq->nr_running == 1 &&
+      rq->lrq.raw_weighted_load > imbalance)
     continue;

-  if (rq->raw_weighted_load > max_load) {
-   max_load = rq->raw_weighted_load;
+  if (rq->lrq.raw_weighted_load > max_load) {
+   max_load = rq->lrq.raw_weighted_load;
     busiest = rq;
   }
  }
@@ -2838,9 +2846,9 @@
  struct rq *rq;

  rq = task_rq_lock(p, &flags);
- ns = p->sum_exec_runtime;
+ ns = p->se.sum_exec_runtime;
  if (rq->curr == p) {
-  delta_exec = rq_clock(rq) - p->exec_start;
+  delta_exec = rq_clock(rq) - p->se.exec_start;
    if ((s64)delta_exec > 0)
     ns += delta_exec;
  }
@@ -3538,7 +3546,7 @@
  rq = task_rq_lock(p, &flags);

  oldprio = p->prio;
- on_rq = p->on_rq;
+ on_rq = p->se.on_rq;
  if (on_rq)
   dequeue_task(rq, p, 0);

@@ -3591,7 +3599,7 @@
   p->static_prio = NICE_TO_PRIO(nice);
   goto out_unlock;
  }
- on_rq = p->on_rq;
+ on_rq = p->se.on_rq;
  if (on_rq) {
   dequeue_task(rq, p, 0);
   dec_raw_weighted_load(rq, p);
@@ -3728,7 +3736,7 @@
 static void
 __setscheduler(struct rq *rq, struct task_struct *p, int policy, int prio)
 {
- BUG_ON(p->on_rq);
```

```
+ BUG_ON(p->se.on_rq);

  p->policy = policy;
  switch (p->policy) {
@@ -3836,7 +3844,7 @@
   spin_unlock_irqrestore(&p->pi_lock, flags);
   goto recheck;
  }
- on_rq = p->on_rq;
+ on_rq = p->se.on_rq;
  if (on_rq)
   deactivate_task(rq, p, 0);
  oldprio = p->prio;
@@ -4490,7 +4498,7 @@
  unsigned long flags;

  __sched_fork(idle);
- idle->exec_start = sched_clock();
+ idle->se.exec_start = sched_clock();

  idle->prio = idle->normal_prio = MAX_PRIO;
  idle->cpus_allowed = cpumask_of_cpu(cpu);
@@ -4633,7 +4641,7 @@
   goto out;

  set_task_cpu(p, dest_cpu);
- if (p->on_rq) {
+ if (p->se.on_rq) {
   deactivate_task(rq_src, p, 0);
   activate_task(rq_dest, p, 0);
   check_preempt_curr(rq_dest, p);
@@ -6100,11 +6108,11 @@
   spin_lock_init(&rq->lock);
   lockdep_set_class(&rq->lock, &rq->rq_lock_key);
   rq->nr_running = 0;
-  rq->tasks_timeline = RB_ROOT;
-  rq->clock = rq->fair_clock = 1;
+  rq->lrq.tasks_timeline = RB_ROOT;
+  rq->clock = rq->lrq.fair_clock = 1;

   for (j = 0; j < CPU_LOAD_IDX_MAX; j++)
-   rq->cpu_load[j] = 0;
+   rq->lrq.cpu_load[j] = 0;
 #ifdef CONFIG_SMP
   rq->sd = NULL;
   rq->active_balance = 0;
@@ -6187,15 +6195,15 @@
  read_lock_irq(&tasklist_lock);
```

```
  do_each_thread(g, p) {
-  p->fair_key = 0;
-  p->wait_runtime = 0;
-  p->wait_start_fair = 0;
-  p->wait_start = 0;
-  p->exec_start = 0;
-  p->sleep_start = 0;
-  p->sleep_start_fair = 0;
-  p->block_start = 0;
-  task_rq(p)->fair_clock = 0;
+  p->se.fair_key = 0;
+  p->se.wait_runtime = 0;
+  p->se.wait_start_fair = 0;
+  p->se.wait_start = 0;
+  p->se.exec_start = 0;
+  p->se.sleep_start = 0;
+  p->se.sleep_start_fair = 0;
+  p->se.block_start = 0;
+  task_rq(p)->lrq.fair_clock = 0;
   task_rq(p)->clock = 0;

   if (!rt_task(p)) {
@@ -6218,7 +6226,7 @@
    goto out_unlock;
 #endif

-  on_rq = p->on_rq;
+  on_rq = p->se.on_rq;
   if (on_rq)
    deactivate_task(task_rq(p), p, 0);
   __setscheduler(rq, p, SCHED_NORMAL, 0);
Index: current/kernel/sched_debug.c
===================================================================
--- current.orig/kernel/sched_debug.c 2007-06-09 15:01:39.000000000 +0530
+++ current/kernel/sched_debug.c 2007-06-09 15:07:16.000000000 +0530
@@ -40,16 +40,16 @@
  SEQ_printf(m, "%15s %5d %15Ld %13Ld %13Ld %9Ld %5d "
        "%15Ld %15Ld %15Ld %15Ld %15Ld\n",
   p->comm, p->pid,
-  (long long)p->fair_key,
-  (long long)(p->fair_key - rq->fair_clock),
-  (long long)p->wait_runtime,
+  (long long)p->se.fair_key,
+  (long long)(p->se.fair_key - rq->lrq.fair_clock),
+  (long long)p->se.wait_runtime,
   (long long)p->nr_switches,
   p->prio,
```

```
-  (long long)p->sum_exec_runtime,
-  (long long)p->sum_wait_runtime,
-  (long long)p->sum_sleep_runtime,
-  (long long)p->wait_runtime_overruns,
-  (long long)p->wait_runtime_underruns);
+  (long long)p->se.sum_exec_runtime,
+  (long long)p->se.sum_wait_runtime,
+  (long long)p->se.sum_sleep_runtime,
+  (long long)p->se.wait_runtime_overruns,
+  (long long)p->se.wait_runtime_underruns);
 }

 static void print_rq(struct seq_file *m, struct rq *rq, u64 now)
@@ -70,7 +70,7 @@
  read_lock_irq(&tasklist_lock);

  do_each_thread(g, p) {
-  if (!p->on_rq)
+  if (!p->se.on_rq)
    continue;

   print_task(m, rq, p, now);
@@ -89,8 +89,8 @@
  spin_lock_irqsave(&rq->lock, flags);
  curr = first_fair(rq);
  while (curr) {
-  p = rb_entry(curr, struct task_struct, run_node);
-  wait_runtime_rq_sum += p->wait_runtime;
+  p = rb_entry(curr, struct task_struct, se.run_node);
+  wait_runtime_rq_sum += p->se.wait_runtime;

   curr = rb_next(curr);
  }
@@ -109,9 +109,9 @@
  SEQ_printf(m, "  .%-22s: %Ld\n", #x, (long long)(rq->x))

  P(nr_running);
- P(raw_weighted_load);
+ P(lrq.raw_weighted_load);
  P(nr_switches);
- P(nr_load_updates);
+ P(lrq.nr_load_updates);
  P(nr_uninterruptible);
  SEQ_printf(m, "  .%-22s: %lu\n", "jiffies", jiffies);
  P(next_balance);
@@ -122,18 +122,18 @@
  P(clock_overflows);
  P(clock_unstable_events);
```

```
  P(clock_max_delta);
- P(fair_clock);
- P(delta_fair_clock);
- P(exec_clock);
- P(delta_exec_clock);
- P(wait_runtime);
- P(wait_runtime_overruns);
- P(wait_runtime_underruns);
- P(cpu_load[0]);
- P(cpu_load[1]);
- P(cpu_load[2]);
- P(cpu_load[3]);
- P(cpu_load[4]);
+ P(lrq.fair_clock);
+ P(lrq.delta_fair_clock);
+ P(lrq.exec_clock);
+ P(lrq.delta_exec_clock);
+ P(lrq.wait_runtime);
+ P(lrq.wait_runtime_overruns);
+ P(lrq.wait_runtime_underruns);
+ P(lrq.cpu_load[0]);
+ P(lrq.cpu_load[1]);
+ P(lrq.cpu_load[2]);
+ P(lrq.cpu_load[3]);
+ P(lrq.cpu_load[4]);
 #undef P
  print_rq_runtime_sum(m, rq);

@@ -205,21 +205,21 @@
 #define P(F) \
  SEQ_printf(m, "%-25s:%20Ld\n", #F, (long long)p->F)

- P(wait_start);
- P(wait_start_fair);
- P(exec_start);
- P(sleep_start);
- P(sleep_start_fair);
- P(block_start);
- P(sleep_max);
- P(block_max);
- P(exec_max);
- P(wait_max);
- P(wait_runtime);
- P(wait_runtime_overruns);
- P(wait_runtime_underruns);
- P(sum_exec_runtime);
- P(load_weight);
+ P(se.wait_start);
```

```
+ P(se.wait_start_fair);
+ P(se.exec_start);
+ P(se.sleep_start);
+ P(se.sleep_start_fair);
+ P(se.block_start);
+ P(se.sleep_max);
+ P(se.block_max);
+ P(se.exec_max);
+ P(se.wait_max);
+ P(se.wait_runtime);
+ P(se.wait_runtime_overruns);
+ P(se.wait_runtime_underruns);
+ P(se.sum_exec_runtime);
+ P(se.load_weight);
  P(policy);
  P(prio);
 #undef P
@@ -235,7 +235,7 @@

 void proc_sched_set_task(struct task_struct *p)
 {
- p->sleep_max = p->block_max = p->exec_max = p->wait_max = 0;
- p->wait_runtime_overruns = p->wait_runtime_underruns = 0;
- p->sum_exec_runtime = 0;
+ p->se.sleep_max = p->se.block_max = p->se.exec_max = p->se.wait_max = 0;
+ p->se.wait_runtime_overruns = p->se.wait_runtime_underruns = 0;
+ p->se.sum_exec_runtime = 0;
 }
Index: current/kernel/sched_fair.c
===================================================================
--- current.orig/kernel/sched_fair.c 2007-06-09 15:01:39.000000000 +0530
+++ current/kernel/sched_fair.c 2007-06-09 15:07:16.000000000 +0530
@@ -51,10 +51,10 @@
  */
 static inline void __enqueue_task_fair(struct rq *rq, struct task_struct *p)
 {
- struct rb_node **link = &rq->tasks_timeline.rb_node;
+ struct rb_node **link = &rq->lrq.tasks_timeline.rb_node;
  struct rb_node *parent = NULL;
  struct task_struct *entry;
- s64 key = p->fair_key;
+ s64 key = p->se.fair_key;
  int leftmost = 1;

  /*
@@ -62,12 +62,12 @@
   */
  while (*link) {
```

```
   parent = *link;
-  entry = rb_entry(parent, struct task_struct, run_node);
+  entry = rb_entry(parent, struct task_struct, se.run_node);
   /*
    * We dont care about collisions. Nodes with
    * the same key stay together.
    */
-  if ((s64)(key - entry->fair_key) < 0) {
+  if ((s64)(key - entry->se.fair_key) < 0) {
    link = &parent->rb_left;
   } else {
    link = &parent->rb_right;
@@ -80,31 +80,31 @@
   * used):
   */
  if (leftmost)
-  rq->rb_leftmost = &p->run_node;
+  rq->lrq.rb_leftmost = &p->se.run_node;

- rb_link_node(&p->run_node, parent, link);
- rb_insert_color(&p->run_node, &rq->tasks_timeline);
+ rb_link_node(&p->se.run_node, parent, link);
+ rb_insert_color(&p->se.run_node, &rq->lrq.tasks_timeline);
 }

 static inline void __dequeue_task_fair(struct rq *rq, struct task_struct *p)
 {
- if (rq->rb_leftmost == &p->run_node)
-  rq->rb_leftmost = NULL;
- rb_erase(&p->run_node, &rq->tasks_timeline);
+ if (rq->lrq.rb_leftmost == &p->se.run_node)
+  rq->lrq.rb_leftmost = NULL;
+ rb_erase(&p->se.run_node, &rq->lrq.tasks_timeline);
 }

 static inline struct rb_node * first_fair(struct rq *rq)
 {
- if (rq->rb_leftmost)
-  return rq->rb_leftmost;
+ if (rq->lrq.rb_leftmost)
+  return rq->lrq.rb_leftmost;
  /* Cache the value returned by rb_first() */
- rq->rb_leftmost = rb_first(&rq->tasks_timeline);
- return rq->rb_leftmost;
+ rq->lrq.rb_leftmost = rb_first(&rq->lrq.tasks_timeline);
+ return rq->lrq.rb_leftmost;
 }
```

```
 static struct task_struct * __pick_next_task_fair(struct rq *rq)
 {
- return rb_entry(first_fair(rq), struct task_struct, run_node);
+ return rb_entry(first_fair(rq), struct task_struct, se.run_node);
 }

 /**************************************************************/
@@ -121,13 +121,13 @@
  /*
   * Negative nice levels get the same granularity as nice-0:
   */
- if (curr->load_weight >= NICE_0_LOAD)
+ if (curr->se.load_weight >= NICE_0_LOAD)
   return granularity;
  /*
   * Positive nice level tasks get linearly finer
   * granularity:
   */
- return curr->load_weight * (s64)(granularity / NICE_0_LOAD);
+ return curr->se.load_weight * (s64)(granularity / NICE_0_LOAD);
 }

 static void limit_wait_runtime(struct rq *rq, struct task_struct *p)
@@ -138,30 +138,30 @@
   * Niced tasks have the same history dynamic range as
   * non-niced tasks:
   */
- if (p->wait_runtime > limit) {
-  p->wait_runtime = limit;
-  p->wait_runtime_overruns++;
-  rq->wait_runtime_overruns++;
- }
- if (p->wait_runtime < -limit) {
-  p->wait_runtime = -limit;
-  p->wait_runtime_underruns++;
-  rq->wait_runtime_underruns++;
+ if (p->se.wait_runtime > limit) {
+  p->se.wait_runtime = limit;
+  p->se.wait_runtime_overruns++;
+  rq->lrq.wait_runtime_overruns++;
+ }
+ if (p->se.wait_runtime < -limit) {
+  p->se.wait_runtime = -limit;
+  p->se.wait_runtime_underruns++;
+  rq->lrq.wait_runtime_underruns++;
  }
 }
```

```
 static void __add_wait_runtime(struct rq *rq, struct task_struct *p, s64 delta)
 {
- p->wait_runtime += delta;
- p->sum_wait_runtime += delta;
+ p->se.wait_runtime += delta;
+ p->se.sum_wait_runtime += delta;
  limit_wait_runtime(rq, p);
 }

 static void add_wait_runtime(struct rq *rq, struct task_struct *p, s64 delta)
 {
- rq->wait_runtime -= p->wait_runtime;
+ rq->lrq.wait_runtime -= p->se.wait_runtime;
  __add_wait_runtime(rq, p, delta);
- rq->wait_runtime += p->wait_runtime;
+ rq->lrq.wait_runtime += p->se.wait_runtime;
 }

 static s64 div64_s(s64 divident, unsigned long divisor)
@@ -185,7 +185,7 @@
  */
 static inline void update_curr(struct rq *rq, u64 now)
 {
- unsigned long load = rq->raw_weighted_load;
+ unsigned long load = rq->lrq.raw_weighted_load;
  u64 delta_exec, delta_fair, delta_mine;
  struct task_struct *curr = rq->curr;

@@ -195,23 +195,23 @@
   * Get the amount of time the current task was running
   * since the last time we changed raw_weighted_load:
   */
- delta_exec = now - curr->exec_start;
+ delta_exec = now - curr->se.exec_start;
  if (unlikely((s64)delta_exec < 0))
   delta_exec = 0;
- if (unlikely(delta_exec > curr->exec_max))
-  curr->exec_max = delta_exec;
+ if (unlikely(delta_exec > curr->se.exec_max))
+  curr->se.exec_max = delta_exec;

- curr->sum_exec_runtime += delta_exec;
- curr->exec_start = now;
- rq->exec_clock += delta_exec;
+ curr->se.sum_exec_runtime += delta_exec;
+ curr->se.exec_start = now;
+ rq->lrq.exec_clock += delta_exec;
```

```
  delta_fair = delta_exec * NICE_0_LOAD;
  delta_fair += load >> 1; /* rounding */
  do_div(delta_fair, load);

  /* Load-balancing accounting. */
- rq->delta_fair_clock += delta_fair;
- rq->delta_exec_clock += delta_exec;
+ rq->lrq.delta_fair_clock += delta_fair;
+ rq->lrq.delta_exec_clock += delta_exec;

  /*
   * Task already marked for preemption, do not burden
@@ -221,11 +221,11 @@
   if (unlikely(test_tsk_thread_flag(curr, TIF_NEED_RESCHED)))
    return;

- delta_mine = delta_exec * curr->load_weight;
+ delta_mine = delta_exec * curr->se.load_weight;
  delta_mine += load >> 1; /* rounding */
  do_div(delta_mine, load);

- rq->fair_clock += delta_fair;
+ rq->lrq.fair_clock += delta_fair;
  /*
   * We executed delta_exec amount of time on the CPU,
   * but we were only entitled to delta_mine amount of
@@ -239,8 +239,8 @@
 static inline void
 update_stats_wait_start(struct rq *rq, struct task_struct *p, u64 now)
 {
- p->wait_start_fair = rq->fair_clock;
- p->wait_start = now;
+ p->se.wait_start_fair = rq->lrq.fair_clock;
+ p->se.wait_start = now;
 }

 /*
@@ -260,21 +260,23 @@
  /*
   * Update the key:
   */
- key = rq->fair_clock;
+ key = rq->lrq.fair_clock;

  /*
   * Optimize the common nice 0 case:
   */
- if (likely(p->load_weight == NICE_0_LOAD))
```

```
-  key -= p->wait_runtime;
+ if (likely(p->se.load_weight == NICE_0_LOAD))
+  key -= p->se.wait_runtime;
  else {
-  if (p->wait_runtime < 0)
-   key -= div64_s(p->wait_runtime * NICE_0_LOAD, p->load_weight);
+  if (p->se.wait_runtime < 0)
+   key -= div64_s(p->se.wait_runtime * NICE_0_LOAD,
+         p->se.load_weight);
   else
-   key -= div64_s(p->wait_runtime * p->load_weight, NICE_0_LOAD);
+   key -= div64_s(p->se.wait_runtime * p->se.load_weight,
+         NICE_0_LOAD);
  }

- p->fair_key = key;
+ p->se.fair_key = key;
 }

 /*
@@ -285,21 +287,21 @@
 {
  s64 delta_fair, delta_wait;

- delta_wait = now - p->wait_start;
- if (unlikely(delta_wait > p->wait_max))
-  p->wait_max = delta_wait;
+ delta_wait = now - p->se.wait_start;
+ if (unlikely(delta_wait > p->se.wait_max))
+  p->se.wait_max = delta_wait;

- if (p->wait_start_fair) {
-  delta_fair = rq->fair_clock - p->wait_start_fair;
+ if (p->se.wait_start_fair) {
+  delta_fair = rq->lrq.fair_clock - p->se.wait_start_fair;

-  if (unlikely(p->load_weight != NICE_0_LOAD))
-   delta_fair = div64_s(delta_fair * p->load_weight,
+  if (unlikely(p->se.load_weight != NICE_0_LOAD))
+   delta_fair = div64_s(delta_fair * p->se.load_weight,
         NICE_0_LOAD);
   add_wait_runtime(rq, p, delta_fair);
  }

- p->wait_start_fair = 0;
- p->wait_start = 0;
+ p->se.wait_start_fair = 0;
+ p->se.wait_start = 0;
```

```
 }

 static inline void
@@ -323,7 +325,7 @@
  /*
   * We are starting a new run period:
   */
- p->exec_start = now;
+ p->se.exec_start = now;
 }

 /*
@@ -332,7 +334,7 @@
 static inline void
 update_stats_curr_end(struct rq *rq, struct task_struct *p, u64 now)
 {
- p->exec_start = 0;
+ p->se.exec_start = 0;
 }

 /*
@@ -362,7 +364,7 @@
    if (curr->sched_class == &fair_sched_class)
    add_wait_runtime(rq, curr, -delta_fair);
  }
- rq->fair_clock -= delta_fair;
+ rq->lrq.fair_clock -= delta_fair;
 }

 /**************************************************************/
@@ -371,25 +373,26 @@

 static void enqueue_sleeper(struct rq *rq, struct task_struct *p)
 {
- unsigned long load = rq->raw_weighted_load;
+ unsigned long load = rq->lrq.raw_weighted_load;
  s64 delta_fair, prev_runtime;

  if (p->policy == SCHED_BATCH || !(sysctl_sched_features & 4))
   goto out;

- delta_fair = rq->fair_clock - p->sleep_start_fair;
+ delta_fair = rq->lrq.fair_clock - p->se.sleep_start_fair;

  /*
   * Fix up delta_fair with the effect of us running
   * during the whole sleep period:
   */
```

```
  if (!(sysctl_sched_features & 32))
-  delta_fair = div64_s(delta_fair * load, load + p->load_weight);
- delta_fair = div64_s(delta_fair * p->load_weight, NICE_0_LOAD);
+  delta_fair = div64_s(delta_fair * load,
+       load + p->se.load_weight);
+ delta_fair = div64_s(delta_fair * p->se.load_weight, NICE_0_LOAD);

- prev_runtime = p->wait_runtime;
+ prev_runtime = p->se.wait_runtime;
  __add_wait_runtime(rq, p, delta_fair);
- delta_fair = p->wait_runtime - prev_runtime;
+ delta_fair = p->se.wait_runtime - prev_runtime;

  /*
   * We move the fair clock back by a load-proportional
@@ -399,9 +402,9 @@
  distribute_fair_add(rq, delta_fair);

 out:
- rq->wait_runtime += p->wait_runtime;
+ rq->lrq.wait_runtime += p->se.wait_runtime;

- p->sleep_start_fair = 0;
+ p->se.sleep_start_fair = 0;
 }

 /*
@@ -420,29 +423,29 @@
  update_curr(rq, now);

  if (wakeup) {
-  if (p->sleep_start) {
-   delta = now - p->sleep_start;
+  if (p->se.sleep_start) {
+   delta = now - p->se.sleep_start;
    if ((s64)delta < 0)
     delta = 0;

-   if (unlikely(delta > p->sleep_max))
-    p->sleep_max = delta;
+   if (unlikely(delta > p->se.sleep_max))
+    p->se.sleep_max = delta;

-   p->sleep_start = 0;
+   p->se.sleep_start = 0;
   }
-  if (p->block_start) {
-   delta = now - p->block_start;
```

```
+  if (p->se.block_start) {
+   delta = now - p->se.block_start;
    if ((s64)delta < 0)
     delta = 0;

-   if (unlikely(delta > p->block_max))
-    p->block_max = delta;
+   if (unlikely(delta > p->se.block_max))
+    p->se.block_max = delta;

-   p->block_start = 0;
+   p->se.block_start = 0;
   }
-  p->sum_sleep_runtime += delta;
+  p->se.sum_sleep_runtime += delta;

-  if (p->sleep_start_fair)
+  if (p->se.sleep_start_fair)
    enqueue_sleeper(rq, p);
  }
  update_stats_enqueue(rq, p, now);
@@ -460,11 +463,11 @@
  update_stats_dequeue(rq, p, now);
  if (sleep) {
   if (p->state & TASK_INTERRUPTIBLE)
-   p->sleep_start = now;
+   p->se.sleep_start = now;
   if (p->state & TASK_UNINTERRUPTIBLE)
-   p->block_start = now;
-  p->sleep_start_fair = rq->fair_clock;
-  rq->wait_runtime -= p->wait_runtime;
+   p->se.block_start = now;
+  p->se.sleep_start_fair = rq->lrq.fair_clock;
+  rq->lrq.wait_runtime -= p->se.wait_runtime;
  }
  __dequeue_task_fair(rq, p);
 }
@@ -486,9 +489,9 @@
   * position within the tree:
   */
  dequeue_task_fair(rq, p, 0, now);
- p->on_rq = 0;
+ p->se.on_rq = 0;
  enqueue_task_fair(rq, p, 0, now);
- p->on_rq = 1;
+ p->se.on_rq = 1;

  /*
```

```
   * yield-to support: if we are on the same runqueue then
@@ -496,9 +499,9 @@
   */
  if (p_to && rq == task_rq(p_to) &&
     p_to->sched_class == &fair_sched_class
-    && p->wait_runtime > 0) {
+    && p->se.wait_runtime > 0) {

-  s64 delta = p->wait_runtime >> 1;
+  s64 delta = p->se.wait_runtime >> 1;

    __add_wait_runtime(rq, p_to, delta);
    __add_wait_runtime(rq, p, -delta);
@@ -519,7 +522,7 @@
 __check_preempt_curr_fair(struct rq *rq, struct task_struct *p,
       struct task_struct *curr, unsigned long granularity)
 {
- s64 __delta = curr->fair_key - p->fair_key;
+ s64 __delta = curr->se.fair_key - p->se.fair_key;

  /*
   * Take scheduling granularity into account - do not
@@ -587,13 +590,13 @@
   * start the wait period:
   */
  if (sysctl_sched_features & 16)  {
-  if (prev->on_rq &&
+  if (prev->se.on_rq &&
     test_tsk_thread_flag(prev, TIF_NEED_RESCHED)) {

     dequeue_task_fair(rq, prev, 0, now);
-    prev->on_rq = 0;
+    prev->se.on_rq = 0;
     enqueue_task_fair(rq, prev, 0, now);
-    prev->on_rq = 1;
+    prev->se.on_rq = 1;
    } else
     update_curr(rq, now);
  } else {
@@ -602,7 +605,7 @@

  update_stats_curr_end(rq, prev, now);

- if (prev->on_rq)
+ if (prev->se.on_rq)
    update_stats_wait_start(rq, prev, now);
 }
```

```
@@ -625,8 +628,8 @@
  if (!curr)
   return NULL;

- p = rb_entry(curr, struct task_struct, run_node);
- rq->rb_load_balance_curr = rb_next(curr);
+ p = rb_entry(curr, struct task_struct, se.run_node);
+ rq->lrq.rb_load_balance_curr = rb_next(curr);

  return p;
 }
@@ -638,7 +641,7 @@

 static struct task_struct * load_balance_next_fair(struct rq *rq)
 {
- return __load_balance_iterator(rq, rq->rb_load_balance_curr);
+ return __load_balance_iterator(rq, rq->lrq.rb_load_balance_curr);
 }

 /*
@@ -654,9 +657,9 @@
   * position within the tree:
   */
  dequeue_task_fair(rq, curr, 0, now);
- curr->on_rq = 0;
+ curr->se.on_rq = 0;
  enqueue_task_fair(rq, curr, 0, now);
- curr->on_rq = 1;
+ curr->se.on_rq = 1;

  /*
   * Reschedule if another task tops the current one.
@@ -689,22 +692,22 @@
   * until it reschedules once. We set up the key so that
   * it will preempt the parent:
   */
- p->fair_key = current->fair_key - niced_granularity(rq->curr,
+ p->se.fair_key = current->se.fair_key - niced_granularity(rq->curr,
       sysctl_sched_granularity) - 1;
  /*
   * The first wait is dominated by the child-runs-first logic,
   * so do not credit it with that waiting time yet:
   */
- p->wait_start_fair = 0;
+ p->se.wait_start_fair = 0;

  /*
   * The statistical average of wait_runtime is about
```

```
   * -granularity/2, so initialize the task with that:
   */
-// p->wait_runtime = -(s64)(sysctl_sched_granularity / 2);
+// p->se.wait_runtime = -(s64)(sysctl_sched_granularity / 2);

  __enqueue_task_fair(rq, p);
- p->on_rq = 1;
+ p->se.on_rq = 1;
  inc_nr_running(p, rq);
 }

Index: current/kernel/sched_rt.c
===================================================================
--- current.orig/kernel/sched_rt.c 2007-06-09 15:01:39.000000000 +0530
+++ current/kernel/sched_rt.c 2007-06-09 15:04:54.000000000 +0530
@@ -15,14 +15,14 @@
  if (!has_rt_policy(curr))
   return;

- delta_exec = now - curr->exec_start;
+ delta_exec = now - curr->se.exec_start;
  if (unlikely((s64)delta_exec < 0))
   delta_exec = 0;
- if (unlikely(delta_exec > curr->exec_max))
-  curr->exec_max = delta_exec;
+ if (unlikely(delta_exec > curr->se.exec_max))
+  curr->se.exec_max = delta_exec;

- curr->sum_exec_runtime += delta_exec;
- curr->exec_start = now;
+ curr->se.sum_exec_runtime += delta_exec;
+ curr->se.exec_start = now;
 }

 static void
@@ -89,7 +89,7 @@
  queue = array->queue + idx;
  next = list_entry(queue->next, struct task_struct, run_list);

- next->exec_start = now;
+ next->se.exec_start = now;

  return next;
 }
@@ -97,7 +97,7 @@
 static void put_prev_task_rt(struct rq *rq, struct task_struct *p, u64 now)
 {
  update_curr_rt(rq, now);
```

```
- p->exec_start = 0;
+ p->se.exec_start = 0;
 }

 /*
```

--
Regards,
vatsa

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