
Subject: [RFC] [PATCH 2/3] Introduce two new structures - struct lrq and sched_entity

Posted by [Srivatsa Vaddagiri](#) on Wed, 23 May 2007 16:54:48 GMT

[View Forum Message](#) <> [Reply to Message](#)

This patch groups together fields used by CFS (for SCHED_NORMAL tasks) in task_struct and runqueue into separate structures so that they can be reused in a later patch.

'struct sched_entity' represents the attributes used by CFS for every schedulable entity (task in this case).

'struct lrq' represents the runqueue used to store schedulable entities (tasks in this case) and to maintain various clocks (ex: fair clock for tasks).

This patch also modifies rest of kernel to reflect these new structures.

Intended effect of this patch is zero on overall functionality of scheduler.

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

fs/proc/array.c		2
include/linux/sched.h		44 +++++--
kernel/exit.c		2
kernel/posix-cpu-timers.c		19 +--
kernel/sched.c		195 ++++++-----
kernel/sched_debug.c		85 ++++++-----
kernel/sched_fair.c		238 ++++++-----

7 files changed, 301 insertions(+), 284 deletions(-)

Index: linux-2.6.21-rc7/fs/proc/array.c

```
=====
--- linux-2.6.21-rc7.orig/fs/proc/array.c 2007-05-23 20:46:40.000000000 +0530
+++ linux-2.6.21-rc7/fs/proc/array.c 2007-05-23 20:48:34.000000000 +0530
@@ -412,7 +412,7 @@
 * Use CFS's precise accounting, if available:
 */
 if (!has_rt_policy(task)) {
- utime = nsec_to_clock_t(task->sum_exec_runtime);
+ utime = nsec_to_clock_t(task->se.sum_exec_runtime);
  stime = 0;
 }
```

Index: linux-2.6.21-rc7/include/linux/sched.h

```
=====
--- linux-2.6.21-rc7.orig/include/linux/sched.h 2007-05-23 20:46:40.000000000 +0530
```

+++ linux-2.6.21-rc7/include/linux/sched.h 2007-05-23 20:48:34.000000000 +0530

@@ -838,6 +838,29 @@

```
void (*task_new) (struct rq *rq, struct task_struct *p);
};
```

/* CFS scheduling entity (task, user etc) statistics fields: */

```
+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;
```

@@ -852,34 +875,15 @@

```
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;
- 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;

```

```

    unsigned int policy;
    cpumask_t cpus_allowed;
Index: linux-2.6.21-rc7/kernel/exit.c

```

```

=====
--- linux-2.6.21-rc7.orig/kernel/exit.c 2007-05-23 20:46:40.000000000 +0530
+++ linux-2.6.21-rc7/kernel/exit.c 2007-05-23 20:48:34.000000000 +0530
@@ -124,7 +124,7 @@
    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: linux-2.6.21-rc7/kernel/posix-cpu-timers.c

```

```

=====
--- linux-2.6.21-rc7.orig/kernel/posix-cpu-timers.c 2007-05-23 20:46:40.000000000 +0530
+++ linux-2.6.21-rc7/kernel/posix-cpu-timers.c 2007-05-23 20:48:34.000000000 +0530
@@ -161,7 +161,8 @@
}
static inline unsigned long long sched_ns(struct task_struct *p)
{
-   return (p == current) ? current_sched_runtime(p) : p->sum_exec_runtime;
+   return (p == current) ? current_sched_runtime(p) :
+       p->se.sum_exec_runtime;
}

int posix_cpu_clock_getres(const clockid_t which_clock, struct timespec *tp)
@@ -249,7 +250,7 @@
    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 +468,7 @@
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 +476,7 @@
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 +537,7 @@
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 +1005,7 @@
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 +1050,7 @@
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 +1209,7 @@
  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 +1301,7 @@

```

```

  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: linux-2.6.21-rc7/kernel/sched.c

```

```

=====

```

```

--- linux-2.6.21-rc7.orig/kernel/sched.c 2007-05-23 20:48:26.000000000 +0530

```

```

+++ linux-2.6.21-rc7/kernel/sched.c 2007-05-23 20:48:34.000000000 +0530

```

```

@@ -114,6 +114,23 @@

```

```

  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, prev_fair_clock;
+ u64 exec_clock, prev_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.
*
@@ -129,16 +146,13 @@
* 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 irq irq;

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
@@ -154,10 +168,6 @@

u64 clock, prev_clock_raw;
s64 clock_max_delta;
- u64 fair_clock, prev_fair_clock;
- u64 exec_clock, prev_exec_clock;
- s64 wait_runtime;
- unsigned long wait_runtime_overruns, wait_runtime_underruns;

unsigned int clock_warps;
unsigned int clock_unstable_events;
@@ -168,10 +178,6 @@
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
@@ -573,13 +579,13 @@
static inline void
inc_raw_weighted_load(struct rq *rq, const struct task_struct *p)
{
- rq->raw_weighted_load += p->load_weight;
+ rq->irq.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)
@@ -605,22 +611,22 @@

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_BATCH tasks get minimal weight:
*/
if (p->policy == SCHED_BATCH) {
- 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)
@@ -629,7 +635,7 @@

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)
@@ -637,7 +643,7 @@

```

```

u64 now = rq_clock(rq);

p->sched_class->dequeue_task(rq, p, sleep, now);
- p->on_rq = 0;
+ p->se.on_rq = 0;
}

/*
@@ -725,7 +731,7 @@
/* 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
@@ -742,18 +748,18 @@
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;

@@ -781,7 +787,7 @@

```

```

    * 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;
}
@@ -812,7 +818,7 @@
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);
@@ -860,9 +866,9 @@
    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);
}

/*
@@ -874,9 +880,9 @@
    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);
}

/*
@@ -887,7 +893,7 @@
    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;
}

```

```

/*
@@ -1118,7 +1124,7 @@
    if (!(old_state & state))
        goto out;

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

    cpu = task_cpu(p);
@@ -1173,11 +1179,11 @@
    * 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) {
+   tl + target_load(cpu, idx) <= tl_per_task) ||
+   100*(tl + p->se.load_weight) <= imbalance*load) {
    /*
        * This domain has SD_WAKE_AFFINE and
        * p is cache cold in this domain, and
@@ -1211,7 +1217,7 @@
    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();
@@ -1275,18 +1281,19 @@
    */
    static void __sched_fork(struct task_struct *p)
    {
- p->wait_start_fair = p->wait_start = p->exec_start = p->last_ran = 0;
- p->sum_exec_runtime = 0;
-
- p->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.wait_start_fair = p->se.wait_start = p->se.exec_start = 0;
+ p->se.last_ran = 0;
+ p->se.sum_exec_runtime = 0;
+
+ p->se.wait_runtime = 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;

```

```

/*
@@ -1357,7 +1364,7 @@
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 {
/*
@@ -1372,7 +1379,7 @@

```

```

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

```

```

/**
@@ -1584,17 +1591,19 @@
unsigned long tmp;
u64 tmp64;

```

```

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

- fair_delta64 = this_rq->fair_clock - this_rq->prev_fair_clock + 1;

```

```

- this_rq->prev_fair_clock = this_rq->fair_clock;
-
- exec_delta64 = this_rq->exec_clock - this_rq->prev_exec_clock + 1;
- this_rq->prev_exec_clock = this_rq->exec_clock;
+ fair_delta64 = this_rq->lrq.fair_clock -
+   this_rq->lrq.prev_fair_clock + 1;
+ this_rq->lrq.prev_fair_clock = this_rq->lrq.fair_clock;
+
+ exec_delta64 = this_rq->lrq.exec_clock -
+   this_rq->lrq.prev_exec_clock + 1;
+ this_rq->lrq.prev_exec_clock = this_rq->lrq.exec_clock;

if (fair_delta64 > (s64)LONG_MAX)
    fair_delta64 = (s64)LONG_MAX;
@@ -1620,10 +1629,10 @@

/* 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;
}
}

@@ -1879,7 +1888,8 @@
    * 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 ||
@@ -1892,7 +1902,7 @@

    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
@@ -1989,7 +1999,7 @@

```

```

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

/*
@@ -2223,11 +2233,12 @@

    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;
    }
}
@@ -2830,7 +2841,7 @@
    unsigned long flags;

    local_irq_save(flags);
-   ns = p->sum_exec_runtime + sched_clock() - p->last_ran;
+   ns = p->se.sum_exec_runtime + sched_clock() - p->se.last_ran;
    local_irq_restore(flags);

    return ns;
@@ -3518,7 +3529,7 @@
    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);

@@ -3571,7 +3582,7 @@
    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);
@@ -3708,7 +3719,7 @@
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) {
@@ -3814,7 +3825,7 @@
        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;
@@ -4468,7 +4479,7 @@
    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);
@@ -4587,7 +4598,7 @@
    if (!cpu_isset(dest_cpu, p->cpus_allowed))
        goto out;

- on_rq = p->on_rq;
+ on_rq = p->se.on_rq;
    if (on_rq)
        deactivate_task(rq_src, p, 0);
    set_task_cpu(p, dest_cpu);
@@ -5986,11 +5997,11 @@
    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->lq.tasks_timeline = RB_ROOT;
+ rq->clock = rq->lq.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;
@@ -6072,15 +6083,15 @@

```

```

    read_lock_irq(&tasklist_lock);
    for_each_process(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)) {
@@ -6103,7 +6114,7 @@
        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: linux-2.6.21-rc7/kernel/sched_debug.c

```

=====
--- linux-2.6.21-rc7.orig/kernel/sched_debug.c 2007-05-23 20:46:40.000000000 +0530
+++ linux-2.6.21-rc7/kernel/sched_debug.c 2007-05-23 20:48:34.000000000 +0530

```

```

@@ -40,15 +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)
@@ -69,7 +70,7 @@

```

```

    curr = first_fair(rq);
    while (curr) {
- p = rb_entry(curr, struct task_struct, run_node);
+ p = rb_entry(curr, struct task_struct, se.run_node);
    print_task(m, rq, p, now);

```

```

    curr = rb_next(curr);
@@ -86,8 +87,8 @@
    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);
}
@@ -106,9 +107,9 @@
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);
@@ -119,18 +120,18 @@
P(clock_unstable_events);
P(clock_max_delta);
rq->clock_max_delta = 0;
- P(fair_clock);
- P(prev_fair_clock);
- P(exec_clock);
- P(prev_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.prev_fair_clock);
+ P(lrq.exec_clock);
+ P(lrq.prev_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);

@@ -190,21 +191,21 @@
#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(last_ran);
- P(wait_runtime);
- P(wait_runtime_overruns);
- P(wait_runtime_underruns);
- P(sum_exec_runtime);
+ 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.last_ran);
+ P(se.wait_runtime);
+ P(se.wait_runtime_overruns);
+ P(se.wait_runtime_underruns);
+ P(se.sum_exec_runtime);
#undef P

```

```

{
@@ -218,7 +219,7 @@

```

```

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: linux-2.6.21-rc7/kernel/sched_fair.c

```

=====
--- linux-2.6.21-rc7.orig/kernel/sched_fair.c 2007-05-23 20:46:40.000000000 +0530
+++ linux-2.6.21-rc7/kernel/sched_fair.c 2007-05-23 20:48:34.000000000 +0530
@@ -55,10 +55,10 @@

```

```

*/
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->lq.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;

/*
@@ -66,12 +66,12 @@
*/
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;
@@ -84,31 +84,31 @@
    * 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);
}

/*****/
@@ -125,24 +125,24 @@
/*
 * 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);
}

unsigned long get_rq_load(struct rq *rq)
{
- unsigned long load = rq->cpu_load[CPU_LOAD_IDX_MAX-1] + 1;
+ unsigned long load = rq->lrq.cpu_load[CPU_LOAD_IDX_MAX-1] + 1;

    if (!(sysctl_sched_load_smoothing & 1))
- return rq->raw_weighted_load;
+ return rq->lrq.raw_weighted_load;

    if (sysctl_sched_load_smoothing & 4)
- load = max(load, rq->raw_weighted_load);
+ load = max(load, rq->lrq.raw_weighted_load);

    return load;
}
@@ -156,31 +156,31 @@
/*
 * Niced tasks have the same history dynamic range as
 * non-niced tasks, but their limits are offset.
 */
- if (p->wait_runtime > nice_limit) {

```

```

- p->wait_runtime = nice_limit;
- p->wait_runtime_overruns++;
- rq->wait_runtime_overruns++;
+ if (p->se.wait_runtime > nice_limit) {
+ p->se.wait_runtime = nice_limit;
+ p->se.wait_runtime_overruns++;
+ rq->lrq.wait_runtime_overruns++;
+ }
  limit = (limit << 1) - nice_limit;
- 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_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;
}

/*
@@ -193,15 +193,15 @@
  struct task_struct *curr = rq->curr;

  if (curr->sched_class != &fair_sched_class || curr == rq->idle
- || !curr->on_rq)
+ || !curr->se.on_rq)
    return;
/*
  * 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;
- if (unlikely(delta_exec > curr->exec_max))
- curr->exec_max = delta_exec;
+ delta_exec = now - curr->se.exec_start;
+ if (unlikely(delta_exec > curr->se.exec_max))
+ curr->se.exec_max = delta_exec;

if (sysctl_sched_load_smoothing & 1) {
    unsigned long load = get_rq_load(rq);
@@ -211,24 +211,24 @@
    do_div(delta_fair, load);
} else {
    delta_fair = delta_exec * NICE_0_LOAD;
- do_div(delta_fair, rq->raw_weighted_load);
+ do_div(delta_fair, rq->lrq.raw_weighted_load);
}

- delta_mine = delta_exec * curr->load_weight;
+ delta_mine = delta_exec * curr->se.load_weight;
do_div(delta_mine, load);
} else {
    delta_fair = delta_exec * NICE_0_LOAD;
- delta_fair += rq->raw_weighted_load >> 1;
- do_div(delta_fair, rq->raw_weighted_load);
+ delta_fair += rq->lrq.raw_weighted_load >> 1;
+ do_div(delta_fair, rq->lrq.raw_weighted_load);

- delta_mine = delta_exec * curr->load_weight;
- delta_mine += rq->raw_weighted_load >> 1;
- do_div(delta_mine, rq->raw_weighted_load);
+ delta_mine = delta_exec * curr->se.load_weight;
+ delta_mine += rq->lrq.raw_weighted_load >> 1;
+ do_div(delta_mine, rq->lrq.raw_weighted_load);
}

- 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;

/*
    * Task already marked for preemption, do not burden
@@ -237,7 +237,7 @@
    if (unlikely(test_tsk_thread_flag(curr, TIF_NEED_RESCHED)))

```

```

goto out_nowait;

- 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
@@ -253,8 +253,8 @@
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;
}

/*
@@ -274,29 +274,29 @@
/*
 * 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 {
- int negative = p->wait_runtime < 0;
+ int negative = p->se.wait_runtime < 0;
u64 tmp;

- if (p->load_weight > NICE_0_LOAD) {
+ if (p->se.load_weight > NICE_0_LOAD) {
/* negative-reniced tasks get helped: */

if (negative) {
- tmp = -p->wait_runtime;
+ tmp = -p->se.wait_runtime;
tmp *= NICE_0_LOAD;
- do_div(tmp, p->load_weight);
+ do_div(tmp, p->se.load_weight);

```

```

    key += tmp;
} else {
-   tmp = p->wait_runtime;
-   tmp *= p->load_weight;
+   tmp = p->se.wait_runtime;
+   tmp *= p->se.load_weight;
    do_div(tmp, NICE_0_LOAD);

    key -= tmp;
@@ -305,16 +305,16 @@
    /* plus-reniced tasks get hurt: */

    if (negative) {
-   tmp = -p->wait_runtime;
+   tmp = -p->se.wait_runtime;

    tmp *= NICE_0_LOAD;
-   do_div(tmp, p->load_weight);
+   do_div(tmp, p->se.load_weight);

    key += tmp;
    } else {
-   tmp = p->wait_runtime;
+   tmp = p->se.wait_runtime;

-   tmp *= p->load_weight;
+   tmp *= p->se.load_weight;
    do_div(tmp, NICE_0_LOAD);

    key -= tmp;
@@ -322,7 +322,7 @@
    }
}

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

/*
@@ -333,20 +333,20 @@
{
    s64 delta_fair, delta_wait;

-   delta_wait = now - p->wait_start;
-   if (unlikely(delta_wait > p->wait_max))
-   p->wait_max = delta_wait;
-
-   if (p->wait_start_fair) {

```

```

- delta_fair = rq->fair_clock - p->wait_start_fair;
- if (unlikely(p->load_weight != NICE_0_LOAD))
- delta_fair = (delta_fair * p->load_weight) /
+ delta_wait = now - p->se.wait_start;
+ if (unlikely(delta_wait > p->se.wait_max))
+ p->se.wait_max = delta_wait;
+
+ if (p->se.wait_start_fair) {
+ delta_fair = rq->lrq.fair_clock - p->se.wait_start_fair;
+ if (unlikely(p->se.load_weight != NICE_0_LOAD))
+ delta_fair = (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
@@ -370,7 +370,7 @@
/*
 * We are starting a new run period:
 */
- p->exec_start = now;
+ p->se.exec_start = now;
}

/*
@@ -381,7 +381,7 @@
{
    update_curr(rq, now);

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

/*****
@@ -396,7 +396,7 @@
    if (!(sysctl_sched_load_smoothing & 16))
        goto out;

- delta_fair = rq->fair_clock - p->sleep_start_fair;
+ delta_fair = rq->lrq.fair_clock - p->se.sleep_start_fair;
    if ((s64)delta_fair < 0)
        delta_fair = 0;

```

```

@@ -406,15 +406,15 @@
    */
    if (sysctl_sched_load_smoothing & 8) {
        delta_fair = delta_fair * load;
-   do_div(delta_fair, load + p->load_weight);
+   do_div(delta_fair, load + p->se.load_weight);
    }

    __add_wait_runtime(rq, p, 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;
    }

/*
@@ -433,29 +433,29 @@
    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);
@@ -473,11 +473,11 @@
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);
}
@@ -509,9 +509,9 @@
/* 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;

/*
* Reschedule if another task tops the current one.
@@ -526,11 +526,11 @@
* yield-to support: if we are on the same runqueue then
* give half of our wait_runtime (if it's positive) to the other task:
*/
- if (p_to && p->wait_runtime > 0) {
- p_to->wait_runtime += p->wait_runtime >> 1;

```

```

- p->wait_runtime >>= 1;
+ if (p_to && p->se.wait_runtime > 0) {
+ p_to->se.wait_runtime += p->se.wait_runtime >> 1;
+ p->se.wait_runtime >>= 1;
+ }
- curr = &p->run_node;
+ curr = &p->se.run_node;
  first = first_fair(rq);
  /*
   * Move this task to the second place in the tree:
@@ -547,25 +547,25 @@
    return;
  }

- p_next = rb_entry(next, struct task_struct, run_node);
+ p_next = rb_entry(next, struct task_struct, se.run_node);
  /*
   * Minimally necessary key value to be the second in the tree:
  */
- yield_key = p_next->fair_key + 1;
+ yield_key = p_next->se.fair_key + 1;

  now = __rq_clock(rq);
  dequeue_task_fair(rq, p, 0, now);
- p->on_rq = 0;
+ p->se.on_rq = 0;

  /*
   * Only update the key if we need to move more backwards
   * than the minimally necessary position to be the second:
  */
- if (p->fair_key < yield_key)
- p->fair_key = yield_key;
+ if (p->se.fair_key < yield_key)
+ p->se.fair_key = yield_key;

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

/*
@@ -575,7 +575,7 @@
__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
@@ -631,7 +631,7 @@
 * If the task is still waiting for the CPU (it just got
 * preempted), start the wait period:
 */
- if (prev->on_rq)
+ if (prev->se.on_rq)
    update_stats_wait_start(rq, prev, now);
}

@@ -654,23 +654,23 @@
    if (!first)
        return NULL;

- p = rb_entry(first, struct task_struct, run_node);
+ p = rb_entry(first, struct task_struct, se.run_node);

- rq->rb_load_balance_curr = rb_next(first);
+ rq->lrq.rb_load_balance_curr = rb_next(first);

    return p;
}

static struct task_struct * load_balance_next_fair(struct rq *rq)
{
- struct rb_node *curr = rq->rb_load_balance_curr;
+ struct rb_node *curr = rq->lrq.rb_load_balance_curr;
    struct task_struct *p;

    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;
}
@@ -688,9 +688,9 @@
 * 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.
@@ -723,16 +723,16 @@
 * 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;

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

```

--
 Regards,
 vatsa

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