ulk7

29 2012

Contents

1

- Understanding Linux Kernel 200A4Emacsorg-mode

2

2.1 - 7.1

- I/O /CPU
- //
- nice(),
- getpriority(), setpriority()
- sched_getschrdule(), sched_setscheduler()
- sched_getpara(), sched_setpara()
- sched_yield()
- sched_set_priority_max()
- sched_rr_get_interval()
- sched_setaffinity()
- sched_getaffinity()

2.1.1 -7.1.1

-
- TIF_NEED_RESCHED
- TASK_RUNNINT

2.1.2 -7.1.2

-

2.2 7.2

- SCHED_FIFOSCHED_RR
- SCHED_NORMAL

2.2.1 -7.2.1

- 100-139
-
- -7.2.1.1
  - (140 -) * 20(<120)
  - (140 -) * 5 (>=120)
- -7.2.1.2
  - 100-139
  -
  - max(100, min(-bouns+5, 130))
  - bouns
    - $\mathrm{TASK_{INTERRUPTE}}$ and $\mathrm{TASK_{UNINTERRUPT}}$ add in difference way, $\mathrm{TASK_{RUNNING}}$ minus.
  -
- -7.2.1.3
  -
  -
- -7.2.1.4

- 1-99,
-
- SCHED_RR

2.3 -7.3

2.3.1 -7.3.1 runqueue

- runqueues
- this_rq(), cpu_rq(n)
- CPU
- arrays
-

2.3.2 -7.3.2

-
- sched_fork()
- sched_clock()

2.4 -7.4

2.4.1 -7.4.1 scheduler_tick()

- timestamp_last_tick
- swap process * TIF_NEED_RESCHED * $\text{hyper}_{\text{threading}}$
- haven't replace? set TIF_NEED_RESCHED, go out
- update time, RT or normal
- lock rq
- unlock rq
- reblance_tick()
- -7.4.1.1

  - FIFO
    * nothing to do
  - RR
    * decrease timeslice
    * moving to the active list tail if timeout

- -7.4.1.2

  - decrease timeslice
  - if timeout
    * however, dequeue_task() from active list
    * set TIF_NEED_RESCHED
    * effective_prio() for getting dynamic prio with avg sleeptime
    * reset timeslice(base on the last step)
    * clean first_time_slice
    * set expired_timestamp if 0
    * insert active or expired
      · insert expried
      · not TASK_INTERACTIVE
      · EXPIRED_STARVING
      · insert active
  - not out
    * TIMESILE_GRANULARITY

2.4.2 -7.4.2 try_to_wake_up()

1. task_rq_lock()
2. $\text{stat}_{\text{mask}}$
3. p->array null
   (a) move to CPU
   (b) $\text{nr}_{\text{uninterruptible}}$, p->actived = -1
   (c) active_task()

i. sched_clock()
ii. recalc_task_prio()
iii. p->actived evaluate 2 or 1
iv. p->stimestamp.
v. insert active list

4. local CPUsync and $\text{TASK}_{\text{PREEMPTSCURR}}$()(task can preempt curr), resched_task(), uni/multiprocessor
5. TASK_RUNNIGN
6. unlock rq

2.4.3 -7.4.3 recalc_task_prio()

- it's a static function
- step:

1. calc avg sleeptime and dynamic prio
2. min(now - p->timestamp, 109)
3. not greater than 0
4. p->$\text{sleep}_{\text{avg}}$ = 900(empirical, max sleep time subtract timeslice), if not thread not $\text{TASK}_{\text{UNIT}}$ and great $\text{INTERACTIVE}_{\text{SLEEP}}$(); go $_{a}$
5. CURRENT_BONUS, $\text{sleep}_{\text{time}}$ mult ($\text{MAX}_{\text{BONUS}}$ - $\text{CURRENT}_{\text{BONUS}}$)
6. is not thread, is $\text{TASK}_{\text{UNINT}}$
7. $\text{sleep}_{\text{time}}$ add to p->$\text{sleep}_{\text{avg}}$
8. must smaller than 1000
9. $_{a}effective$prio

- rewrite

```
static void recalc_task_prio(task_t *p, unsigned long long now)

{

    /* Caller must always ensure 'now >= p->timestamp' */

    unsigned long long __sleep_time = now - p->timestamp;

    unsigned long sleep_time;


    if (__sleep_time > NS_MAX_SLEEP_AVG)

        sleep_time = NS_MAX_SLEEP_AVG;
    else
        sleep_time = (unsigned long)__sleep_time;

    if (likely(sleep_time > 0)) {
        /* normal, TASK_UNINTERRUPT */
        if (p->mm && p->activated == -1){
            sleep_time *= (MAX_BONUS - CURRENT_BONUS(p)) ? : 1;

            if (p->sleep_avg >= INTERACTIVE_SLEEP(p)){
                sleep_time = 0;
            }
            else if (p->sleep_avg + sleep_time >=
                     INTERACTIVE_SLEEP(p)) {
                p->sleep_avg = INTERACTIVE_SLEEP(p);
                sleep_time = 0;
            }

            p->sleep_avg += sleep_time;
            if (p->sleep_avg > NS_MAX_SLEEP_AVG)
                p->sleep_avg = NS_MAX_SLEEP_AVG;

        }
        /* normal, not TASK_UNINTERRUPT */
        else if (p->mm && p->activated != -1)
        {
            if (sleep_time > INTERACTIVE_SLEEP(p)){
```

```
                p->sleep_avg = JIFFIES_TO_NS(MAX_SLEEP_AVG -
                        DEF_TIMESLICE);
            }
            else{
                sleep_time *= (MAX_BONUS - CURRENT_BONUS(p)) ? : 1;

                p->sleep_avg += sleep_time;
                if (p->sleep_avg > NS_MAX_SLEEP_AVG)
                    p->sleep_avg = NS_MAX_SLEEP_AVG;

            }
        }else{      /* thread (!p->mm) and other */
            sleep_time *= (MAX_BONUS - CURRENT_BONUS(p)) ? : 1;

            p->sleep_avg += sleep_time;
            if (p->sleep_avg > NS_MAX_SLEEP_AVG)
                p->sleep_avg = NS_MAX_SLEEP_AVG;
        }

        p->prio = effective_prio(p);
    }
}
```

2.4.4 -7.4.4 schedule()

- -7.4.4.1 direct invocation

  - for resource
  - 5 steps

    1. insert wait list
    2. TASK_{UN}INTERRUPTIBLE
    3. schedule()
    4. check resource
    5. remove from list

- -7.4.4.2 lazy invocation

- TIF_NEED_RESCHED
- example
  1. $\text{scheduler}_{\text{tick}}()$
  2. $\text{try}_{\text{towakeup}}()$
  3. $\text{sched}_{\text{setschedule}}()$

- -7.4.4.3 actions performed by schedule() before a process switch

  1. in exiting and in atomic then dump
  2. $\text{prifile}_{\text{hit}}()$
  3. $\text{preempt}_{\text{disable}}()$, $\text{release}_{\text{kernellock}}()$, $\text{this}_{\text{rq}}()$
  4. it's idle thread and not in running then $\text{dump}_{\text{stack}}()$;
  5. check kernel lock
  6. idle thread is not allowed to schedule, $\text{dump}_{\text{stack}}()$
  7. get run_{time}, $\text{sched}_{\text{clock}}()$-prev->timestamp
  8. limit in 1s
  9. lock rq
  10. PF_DEAD
  11. not in running stat and not be preempt in kernel mode then remove from rq
  12. $\text{TASK}_{\text{INTERRUPTIBLE}}$(no $\text{TASK}_{\text{STOPPED}}$) and not pending by signal then set RUNNING, and it will also be the next.
  13. idle_balance()
  14. active <-> expired
  15. bitmask
  16. add sleeptime then reinster to rq->active

- $\text{TASK}_{\text{INTERRUPTIBLE}}$ or $\text{TASK}_{\text{STOPPED}}$
  (a) by system call
  (b) by interrupt or deferred function

- -7.4.4.4

1. prefetch
2. clear next's TIF_NEED_RESCHED
3. $\text{rcq}_{\text{qsctrinc}}$
4. minus next't sleeptime, timestamps
5. prev == next
6. $\text{active}_{\text{mm}}$(using) and mm(own) field.
7. prev is kernel thread or a exit process
   – set prev_mm field

- -7.4.4.5 schedule()

1. barrier()
2. finish_task_switch()
   (a) unlock rq, enable irq
   (b) put_task_struct() if prev is zombie
3. kernel lock, enabel preempt, check TIF_NEED_RESCHED

2.5 -7.4

- flavous
- NUMA
-

2.5.1 -7.5.1

- CPU
-
- sched_domain, sched_group, groups, parent
- phys_domains, sd

2.5.2 -7.5.2 rebalance_tick()

- scheduler_tick
- 3
- cpu_load
- load_balance(),

2.5.3 -7.5.3 load_balance()

-
- find_busest_group()
-
- find_busiest_queue(),
- move_tasks()
-
- active_balance, migrtion_thread
-

2.5.4 -7.5.4 move_tasks()

- NEWLY_IDLE
- expired,
- active can_migrate_task()
- CPUcpus,allowed,idle,,"cache hot"
- pull_task(), dequeue/enqueue_task(), resched_task

2.6 -7.6

2.6.1 -7.6.1 nice()

- sys_nice()
- 40
- capable()
- security_task_setnice()
- static_prio
- setuser_nice()
- resched_task()

2.6.2 -7.6.2 getpriority() setpriority()

- 20
- PRIO_PROCESS/PGRP/USER

2.6.3 -7.6.3 sched_get(SET)AFFINITY()

- cpus_allows,
-

2.6.4 -7.6.4

- -7.6.4.1 sched_get(set)scheduler()
  - sys_sched_getschedule()
  - policy
  - do_sched_setscheduler()
  -
- -7.6.4.2 sched_get(set)param()

  - rt_priority
  - expiredrunqueue

- -7.6.4.3 sched_yield()

  - expiredrunqueue

- -7.6.4.4

- -7.6.4.5 sched_rr_get_interval()

  -
  -
  - FIFO

3 other

3.1 effective$_{\text{prio}}$()

- the dynamic prio of process(rt or normal) get from this function.
- if it's rt process , just return the dynamic prio without bonuse and penalty
- formula for get current bonus : current bonus / $\text{MAX}_{\text{BONUS}}$ = current $\text{sleep}_{\text{avg}}$ / $\text{MAX}_{\text{SLEEPAVG}}$
- formala for get $\text{MAX}_{\text{BONUS}}$: $\text{MAX}_{\text{BONUS}}$ / $\text{MAX}_{\text{USERPRIO}}$ = $\text{PRIO}_{\text{BONUSRATIO}}$ / 100
- the dynamic prio always get with static prio subtract current bonus.
- $\text{USER}_{\text{PRIO}}$ macro does not include the rt, so it is MAX_{PRIO} subtract $\text{MAX}_{\text{RTPRIO}}$,
- there is an express in ulk: and effective$_{\text{prio}}$ has a code block:
  so $\text{MAX}_{\text{BONUS}}$ is 10, $\text{CURRENT}_{\text{BONUS}}$(p) is between 0 and 10.

3.2 NICE AND PRIO

```
/*
 * Convert user-nice values [ -20 ... 0 ... 19 ]
 * to static priority [ MAX_RT_PRIO..MAX_PRIO-1 ],
 * and back.
 */
#define NICE_TO_PRIO(nice) (MAX_RT_PRIO + (nice) + 20)
#define PRIO_TO_NICE(prio) ((prio) - MAX_RT_PRIO - 20)
#define TASK_NICE(p)       PRIO_TO_NICE((p)->static_prio)
```

- we can learn that when prio increase by 1 , nice increase by 1.
- start form $\mathrm{MAX}_{\mathrm{RTPRIO}}$.
- relate to the static prio, not dynamic prio
- the rt task's nice is smaller than -20.

```
/*
 * 'User priority' is the nice value converted to something we
 * can work with better when scaling various scheduler parameters,
 * it's a [ 0 ... 39 ] range.
 */
#define USER_PRIO(p)        ((p)-MAX_RT_PRIO)
#define TASK_USER_PRIO(p)   USER_PRIO((p)->static_prio)
#define MAX_USER_PRIO       (USER_PRIO(MAX_PRIO))
```

- min timeslice 5ms, default 100ms, max 800ms

```
/*
 * These are the 'tuning knobs' of the scheduler:
 *
 * Minimum timeslice is 5 msecs (or 1 jiffy, whichever is larger),
 * default timeslice is 100 msecs, maximum timeslice is 800 msecs.
 * Timeslices get refilled after they expire.
 */
#define MIN_TIMESLICE      max(5 * HZ / 1000, 1)
#define DEF_TIMESLICE      (100 * HZ / 1000)
#define ON_RUNQUEUE_WEIGHT  30
#define CHILD_PENALTY       95
#define PARENT_PENALTY     100
```

```
#define EXIT_WEIGHT          3
#define PRIO_BONUS_RATIO    25
#define MAX_BONUS      (MAX_USER_PRIO * PRIO_BONUS_RATIO / 100)
#define INTERACTIVE_DELTA    2
#define MAX_SLEEP_AVG      (DEF_TIMESLICE * MAX_BONUS)
#define STARVATION_LIMIT   (MAX_SLEEP_AVG)
#define NS_MAX_SLEEP_AVG   (JIFFIES_TO_NS(MAX_SLEEP_AVG))
```

- one tick, jiffies increase one, 100HZ means that 1s 100tick
  - Linuxtimer interrupt (IRQ 0)HZ timer interruptsHZ10001000 timer interruptsHZ

    http://adrianhuang.blogspot.com/2007/10/linux-kernel-hz-tick-and-jiffies.html
    - TickHZtimer interruptHZ 250tick4 (millisecond)
    - jiffiesLinux(32unsigned long) ticktimer interrupt Jiffies
- sched.cshow$_{\text{schedstat}}$()/proc/schedstat
- SCHEDSTAT$_{\text{VERSION}}$
- yld$_{\text{bothempty}}$?
- yld$_{\text{actempty}}$? yld$_{\text{expempty}}$?yld$_{\text{cnt}}$?

4 all function in sched.c

4.1 task$_{\text{rqunlock}}$()