Linaro connect
San Francisco 2017

SFO17- 421 The Linux Kernel Scheduler

Viresh Kumar (PMWG)

Topics

- CPU Scheduler
- The O(1) scheduler
- Current scheduler design
- Scheduling classes
- schedule()
- Scheduling classes and policies
- Sched class: STOP
- Sched class: Deadline
- Sched class: Realtime
- Sched class: CFS
- Sched class: Idle
- Runqueue

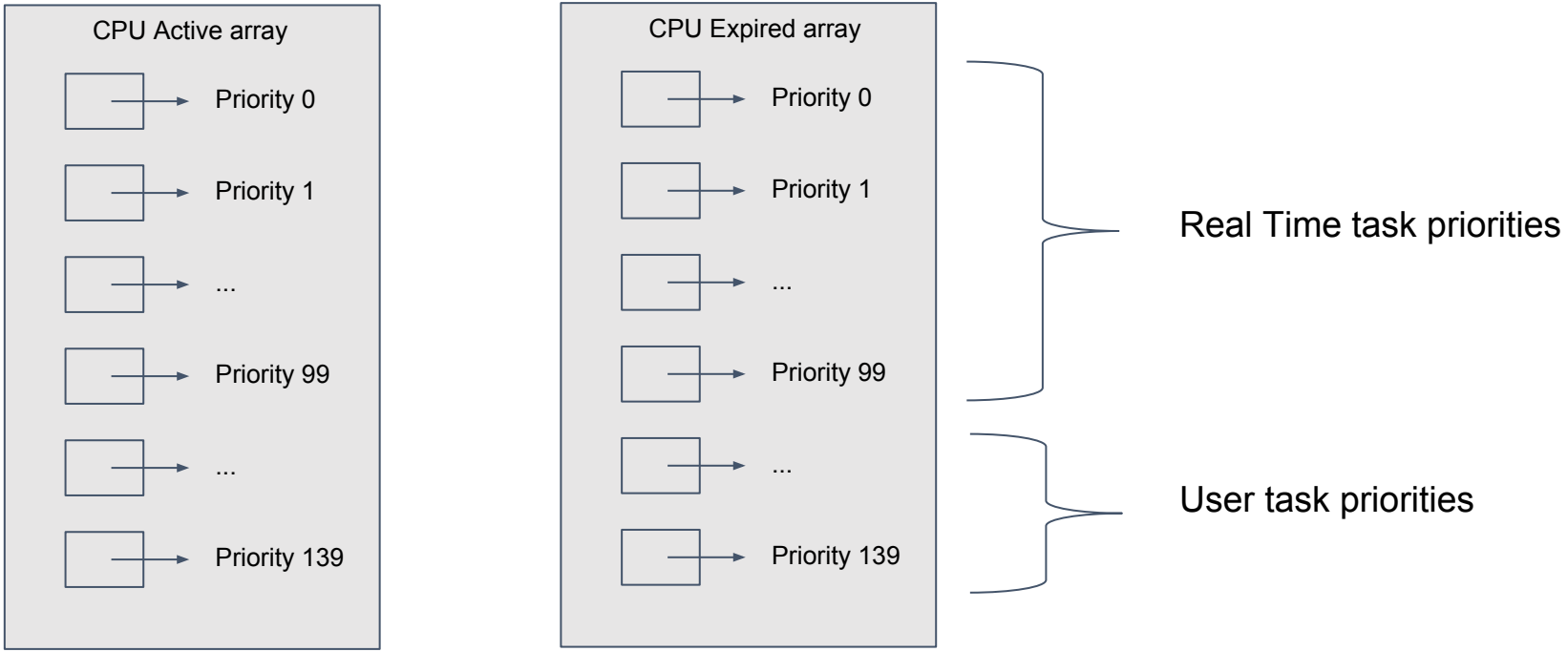
CPU Scheduler

- Shares CPU between tasks.
- Selects next task to run (on each CPU) when:
  - Running task terminates
  - Running task sleeps (waiting for an event)
  - A new task created or sleeping task wakes up
  - Running task's timeslice is over
- Goals:
  - Be *fair*
  - Timeslice based on task priority
  - Low task response time
  - High (task completion) throughput
  - Balance load among CPUs
  - Power efficient
  - Low overhead of running scheduler's code
- Work with mainframes, servers, desktops/laptops, embedded/phones.

The O(1) scheduler

- 140 priorities. 0-99: RT tasks and 100-139: User tasks.
- Each CPU’s runqueue had 2 arrays: Active and Expired.
- Each arrays had 140 entries, one per priority level.
- Each entry was a linked list serviced in FIFO order.
- Bitmap (of 140 bits) used to find which priority lists aren’t empty.
- Timeslices were assigned to tasks based on these priorities.
- Expired tasks moved from Active to Passive array.
- Once Active array was empty, the arrays were swapped.
- Enqueue and dequeue of tasks and next task selection done in constant time.
- Replaced by CFS in Linux 2.6.23 (2007).

The O(1) scheduler (Cont..)



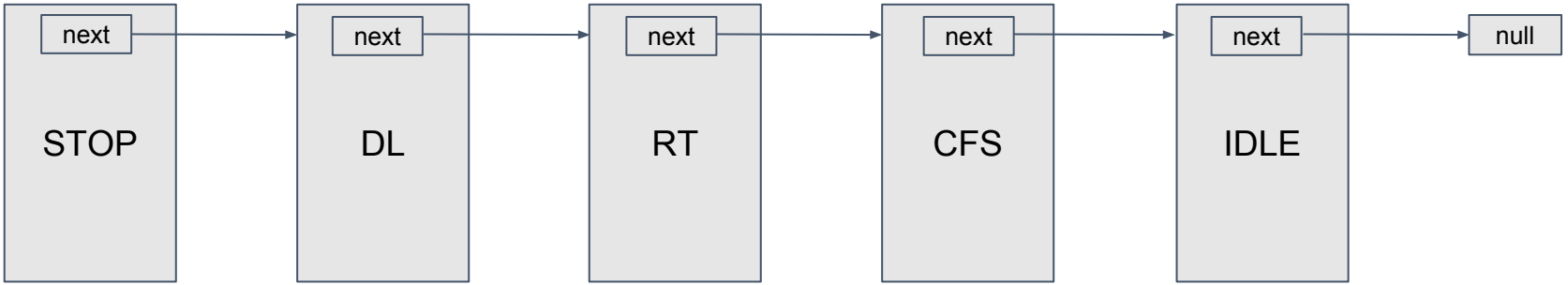
Current scheduler design

- Introduced by Ingo Molnar in 2.6.23 (2007).
- Scheduling policies within scheduling classes.
- Scheduling class with higher priority served first.
- Task can migrate between CPUs, scheduling policies and scheduling classes.

Scheduling Classes

- Represented by following structure:

```
struct sched_class {
        const struct sched_class *next;
        void (*enqueue_task) (struct rq *rq, struct task_struct *p, int flags);
        void (*dequeue_task) (struct rq *rq, struct task_struct *p, int flags);
        struct task_struct * (*pick_next_task) (struct rq *rq, struct task_struct *prev, struct rq_flags *rf);
        ...
};
```



Schedule()

- Picks the next runnable task to run on a CPU.
- Searches for a task starting from the highest priority class (stop).
- Helper: for_each_class().
- pick_next_task():

```
again:
        for_each_class(class) {
                p = class->pick_next_task(rq, prev, rf);
                if (p) {
                        if (unlikely(p == RETRY_TASK))
                                goto again;
                        return p;
                }
        }

        /* The idle class should always have a runnable task: */
        BUG();
```

Scheduling classes and policies

- Stop
  - No policy
- Deadline
  - SCHED_DEADLINE
- Real Time
  - SCHED_FIFO
  - SCHED_RR
- Fair
  - SCHED_NORMAL
  - SCHED_BATCH
  - SCHED_IDLE
- Idle
  - No policy

Sched class: STOP

- Highest priority class.
- Only available for SMP (stop_machine() isn't used in UP).
- Can preempt everything and is preempted by nothing.
- Mechanism to stop running everything else and run a specific function on CPU(s).
- No scheduling policies.
- One kernel thread per CPU: "migration/N".
- Used by task migration, CPU hotplug, RCU, ftrace, clockevents, etc.

Sched class: Deadline (DL)

- Introduced by Dario Faggioli & Juri Lelli, v3.14 (2013).
- Highest priority tasks in the system.
- Scheduling policy: SCHED_DEADLINE.
- Implemented with red-black tree (self balancing).
- Used for periodic real time tasks, eg. Video encoding/decoding.

Sched class: Realtime (RT)

- POSIX “real-time” tasks.
- Task priorities: 0 - 99.
- Inverted priorities: 0 highest in kernel, lowest in user space.
- Scheduling policies for tasks at same priority:
  - SCHED_FIFO
  - SCHED_RR, 100ms timeslice by default.
- Implemented with Linked lists.
- Used for short latency sensitive tasks, eg. IRQ threads.

Sched class: CFS (Completely fair scheduler)

- Introduced by Ingo Molnar (motivated by Rotating Staircase Deadline Scheduler by Con Kolivas).
- Scheduling policies:
  - SCHED_NORMAL: Normal Unix tasks
  - SCHED_BATCH: Batch (non-interactive) tasks
  - SCHED_IDLE: Low priority tasks
- Implemented with red-black tree (self balancing).
- Tracks Virtual runtime of tasks (amount of time task has run).
- Tasks with shortest vruntime runs first.
- Priority is used to set task's weight, that affects vruntime.
- Higher the weight, slower will vruntime increase.
- Task's priority is calculated by 120 + nice (-20 to +19).
- Used for all other system tasks, eg: shell.

Sched class: Idle

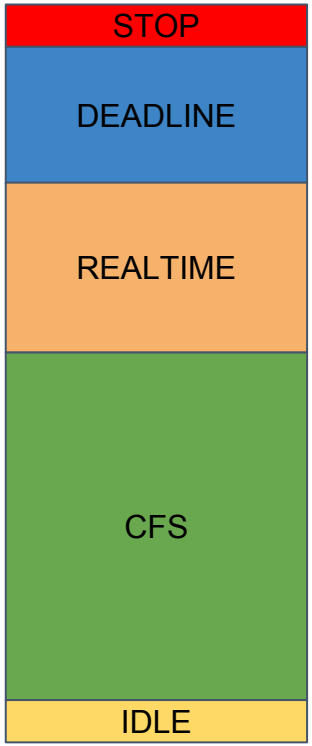
- Lowest priority scheduling class.
- No scheduling policies.
- One kernel thread (idle) per CPU: “swapper/N”.
- Idle thread runs only when nothing else is runnable on a CPU.
- Idle thread may take the CPU to low power state.

Runqueue

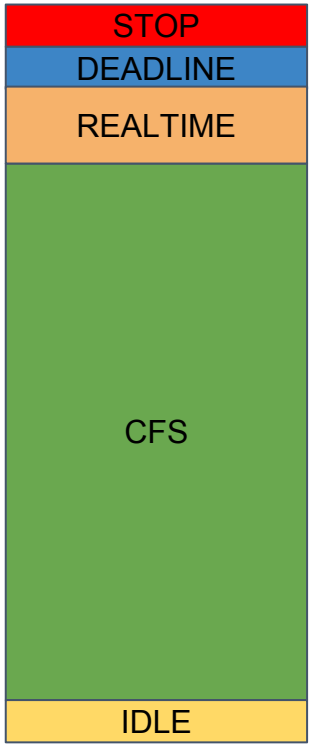
- Each CPU has an instance of “struct rq”.
- Each “rq” has DL, RT and CFS runqueues within it.
- Runnable tasks are enqueued on those runqueues.
- Lots of other information, stats are available in struct rq.

```
struct rq {
        …
        struct cfs_rq cfs;
        struct rt_rq rt;
        struct dl_rq dl;
        ...
}
```

Runqueue (Cont.)



RQ CPU 0



RQ CPU 1

Linaro connect
San Francisco 2017

Thank You

#SFO17

BUD17 keynotes and videos on: connect.linaro.org

For further information: www.linaro.org