Linux Scheduler Implementation

- Scheduler is responsible for selecting next process for its execution.
- The scheduler of Linux is implmented in different ways in its lifetime.

O(n) scheduler

- Developed by Linus Torvalds and is based on UNIX scheduler.
- It go through all processes to pick up next process for the execution.
- The time complexity for scheduler is O(n).
- This is very slow if number of process are too many.

O(1) scheduler

- Developed by Ingo Monlar and is designed for realtime behaviour.
- The time required for picking next process is irrespective of number of processes.
- Ready queue is divided into multi-level queue. Number of levels is same as number of priorities. A ready process with highest priority is selected from active ready queue. Upon completion the process is moved to the expired ready queue. When all processes from active ready queue are completed, active and expired ready queues are interchanged.
- This scheduler is introduced in Linux kernel 2.5 and was used till Linux kernel 2.6.23. However this scheduler is still preferred in realtime Linux.

CFS scheduler

- Completely fair scheduler introduced in Linux kernel 2.6.24, to give fair amount of CPU time for each process.
- CFS scheduler is internally implemented using RB tree (which is a kind of self-balancing binary search tree). The processes are arranged in tree on basis of vruntime (Virtual Runtime).
- The time required for picking next process is O(log n)

Linux Scheduling

- Scheduling policies
  - Realtime policies
  - Non-realtime policies
- Scheduler implementations
  - O(1) scheduler
  - CFS scheduler

Linux Scheduling policies/classes

- terminal> man 7 sched
- There are two categories of scheduling policies:
  - Realtime scheduling classes
    - SCHED_FIFO
    - SCHED_RR
  - Non-realtime scheduling classes

- SCHED_OTHER or SCHED_NORMAL
- SCHED_BATCH
- SCHED_IDLE

Realtime vs Non-realtime scheduling

- In realtime sched classes, each process have priority from 0 (highest) to 99 (lowest) -- called as realtime priority (rtprio).
- In realtime sched classes, a high priority process doesn't wait/block for a low priority process.
- Linux follows soft-realtime scheduling.

```
ps -e -o pid,cls,rtprio,cmd
```

- In non-realtime sched classes, each process have nice value from -20 (highest) to +19 (lowest).
- In non-realtime sched classes, higher priority process will get more CPU time share for its execution, while lower priority process will get less CPU time share for its execution.
- Default scheduling policy of Linux is SCHED_OTHER (non-realtime sched class).

```
ps -e -o pid,cls,ni,cmd
```

Linux Task Priorities

- Realtime priorities -- 0 (highest) to 99 (lowest)
- Non-realtime priorities -- -20 (highest) to +19 (lowest)
- Real-time priorities are always higher than non-realtime priorities. For this non-realtime priorities (-20 to +19) are mapped to 100 to 139.
- terminal> ps -e -o pid,cls,rtprio,ni,pri,cmd
- The output of PRI is inverted i.e. 0 is lowest and 139 is highest priority.

Changing scheduling classes & priorities

- In Linux, child process inherit sched class and priority of parent process.
- chrt command is used to modify scheduling class and priority of existing process or new process.
- To assign realtime sched class, process must have root privilege.
- chrt command internally call sched_setscheduler() syscall to change sched class and priority.

```
sudo chrt -f 20 ./sched.out
sudo chrt -r 20 ./sched.out
sudo chrt -o 110 ./sched.out
```

- renice & nice commands are used to modify nice value of existing process or new process.

- To increase the nice value, root permission is not needed; but to decrease the decrease the nice value, root permission is required.
- nice command internally call nice() syscall.

```
nice -n 12 ./sched.out
sudo nice -n -3 ./sched.out
sudo renice -n 18 -p <pid>
```

- The non-realtime priorities (i.e. nice values) are always considered as lower priorities than real time priorities.
- Inside kernel, nice values are mapped to 100 to 139, which is after realtime priorities 0 to 99.
- However output of ps command is inverted i.e. 0 is lowest and 139 is highest

```
ps -e -o pid,cls,rtprio,ni,pri,cmd
```

SCHED_FIFO

- SCHED_FIFO scheduling class -- "Realtime" Non-preemptive
  - The process cannot be prempted by any other process with same or lower priority.
  - High priority process can always preempt low priority process.
- Current task stop/pause only (Scheduler is invoked to schedule the next task):
  - When running task is terminated.
    - Running state --> Terminated state
  - When running task request IO / blocking (sync) / sleep.
    - Running state --> Blocked state
  - When running task gives up CPU volunteerily i.e. sched_yield().
    - Running state --> Ready state
    - Co-operative scheduling
  - When high priority task is ready to execute -- Preemption due to "Realtime" scheduling.
    - (Current Low priority process) Running state --> Ready state
- To run any process with given sched class use chrt command.
  - terminal> sudo chrt -f 50 ./a.out
    - -f = SCHED_FIFO.
  - To change sched class of existing process.
    - terminal> sudo chrt -f 50 -p 1234
      - -f = SCHED_FIFO.
      - pid = 1234
  - To change sched class/priority of current process use sched_setscheduler() or sched_setparam().
    - scheduling class/policy = "SCHED_FIFO", SCHED_RR, SCHED_OTHER/SCHED_NORMAL, SCHED_BATCH, SCHED_IDLE.

SCHED_RR

- SCHED_RR -- Real-time Pre-emptive.

    - High priority process can always preempt low priority process.
    - When time slice of the process is completed, then process is forcibly preempted.
- Current task stop/pause only (Scheduler is invoked to schedule the next task):
    - When running task is terminated.
    - When running task request IO / blocking (sync) / sleep.
    - When running task gives up CPU volunteerily i.e. sched_yield().
    - When high priority task is ready to execute --> Preemption (due to high priority process -- realtime)
    - When time slice of current task is completed --> Preemption (with same priority process).
        - (Current process) Running state --> Ready state
- To run any process with given sched class use chrt command.
    - terminal> sudo chrt -r 50 ./a.out
        - -r = SCHED_RR.
- To change sched class of existing process.
    - terminal> sudo chrt -r 50 -p 1234
        - -r = SCHED_RR.
        - pid = 1234
- For chrt command, if no sched class is given then default sched class = SCHED_RR.
- To change sched class/priority of current process use sched_setscheduler() or sched_setparam().
    - scheduling class/policy = SCHED_FIFO, "SCHED_RR", SCHED_OTHER/SCHED_NORMAL, SCHED_BATCH, SCHED_IDLE.
- RR time-slice can be seen/modified via /proc/sys/kernel/sched_rr_timeslice_ms.

SCHED_OTHER

- Linux follows time-share (TS) scheduling methodology for SCHED_OTHER.
- In this scheduling, CPU time is divided into epoch times (also called as targetted latency) e.g. 100 ms. All processes ready for execution will be given some CPU time for execution within an epoch.
- The time is divided among ready processes based on nice value of the process. More is the nice value, less is the time alloted to the process.
- The time share for each process is recalculated at the beginning of new epoch.
- A weight (Wi) is pre-computed for each nice value.
    - Less the nice value, more is the weight and hence more is CPU time.
    - More the nice value, less is the weight and hence less is CPU time.
    - The difference in weights of two consecutive nice values is mapped to 10% of CPU time.
- The CPU time given for each process = (Wi / (W0 + W1 + W2 + ...)) * epoch time.