Scheduling in Linux

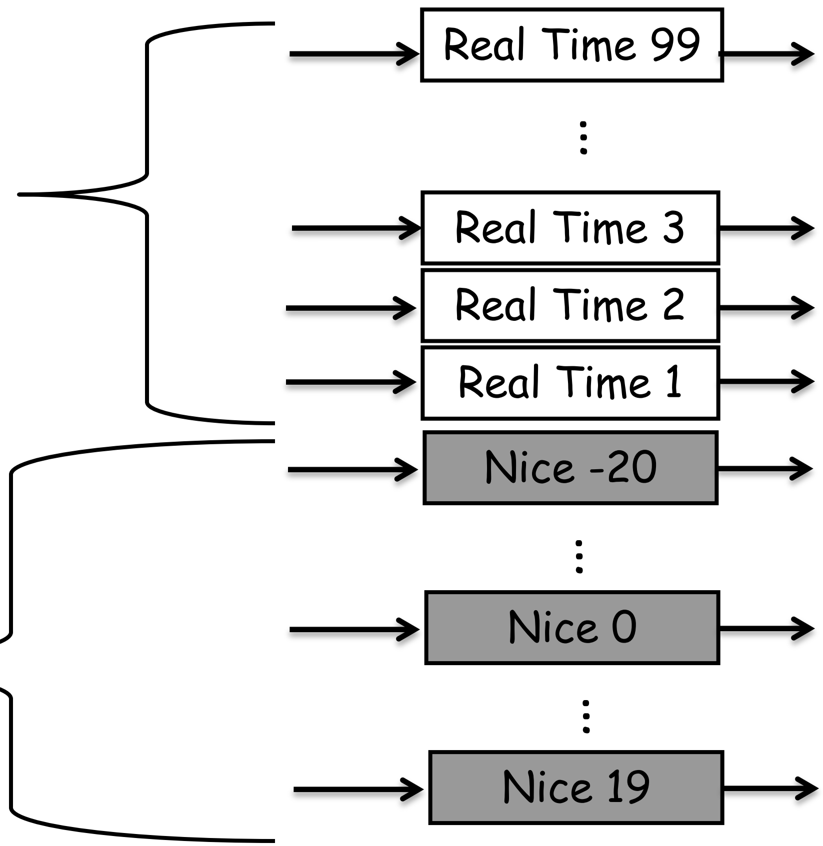
Real-time scheduling

- Hard real-time
  - complete critical task within guaranteed time period
- Soft real-time
  - critical processes have priority over others

- Linux supports soft real-time

Linux: multi-level queue with priorities

- Soft real-time scheduling policies
  - SCHED_FIFO (FCFS)
  - SCHED_RR (round robin)
  - Priority over normal tasks
  - 100 static priority levels (1..99)
- Normal scheduling policies
  - SCHED_NORMAL: standard
    - SCHED_OTHER in POSIX
  - SCHED_BATCH: CPU bound
  - SCHED_IDLE: lower priority
  - Static priority is 0
    - 40 dynamic priority
    - "Nice" values
- sched_setscheduler(), nice()
- See "man 7 sched" for detailed overview

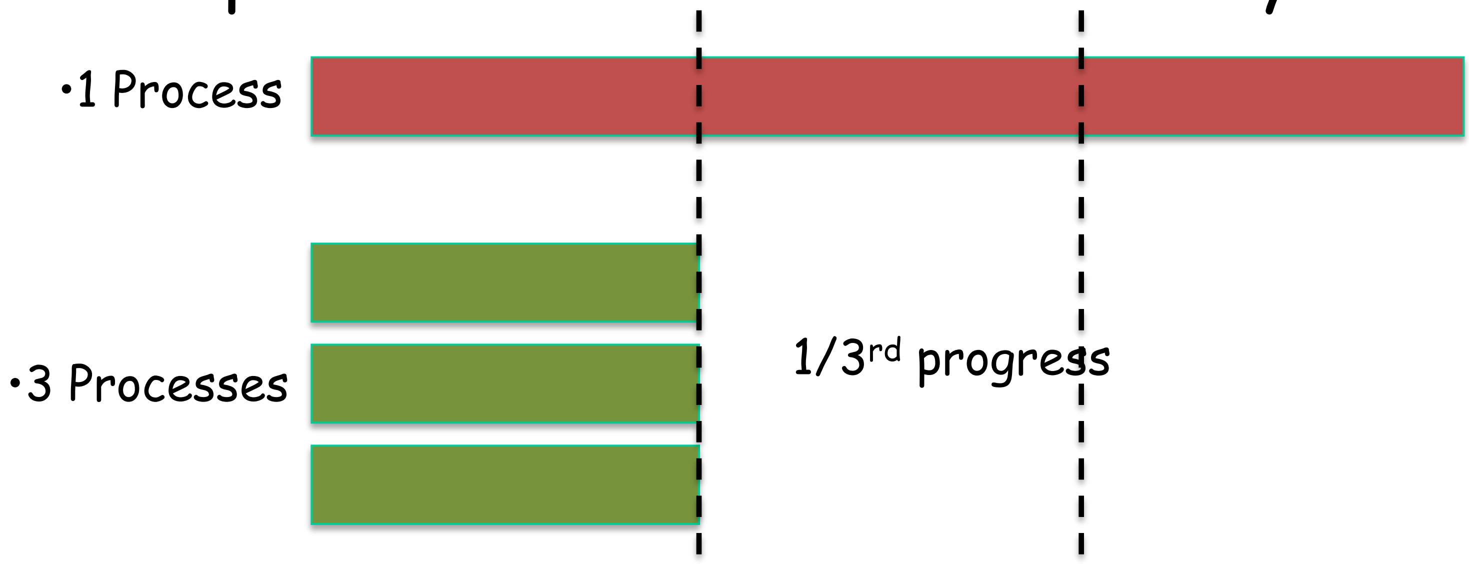


Linux scheduler history

- O(N) scheduler up to 2.4
  - Simple: global run queue
  - Poor performance on multiprocessor and large N
- O(1) scheduler in 2.5 & 2.6
  - Good performance: per-CPU run queue
  - Complex and error prone logic to boost interactivity
  - No fairness guarantee
- Completely Fair Scheduler (CFS) in 2.6 and later
  - Currently default scheduler for SCHED_NORMAL
  - Processes get fair share of CPU
  - Naturally boosts interactivity
- Alternative schedulers: BFS, MuQSS, PDS, BMQ, TT, etc.
  - https://wiki.archlinux.org/title/improving_performance#Alternative_CPU_schedulers

Ideal fair scheduling

- Infinitesimally small time slice
- n processes: each runs uniformly at $1/n^{th}$ rate



- Various approximations of the ideal
  - Lottery scheduling
  - Stride scheduling
  - Linux CFS

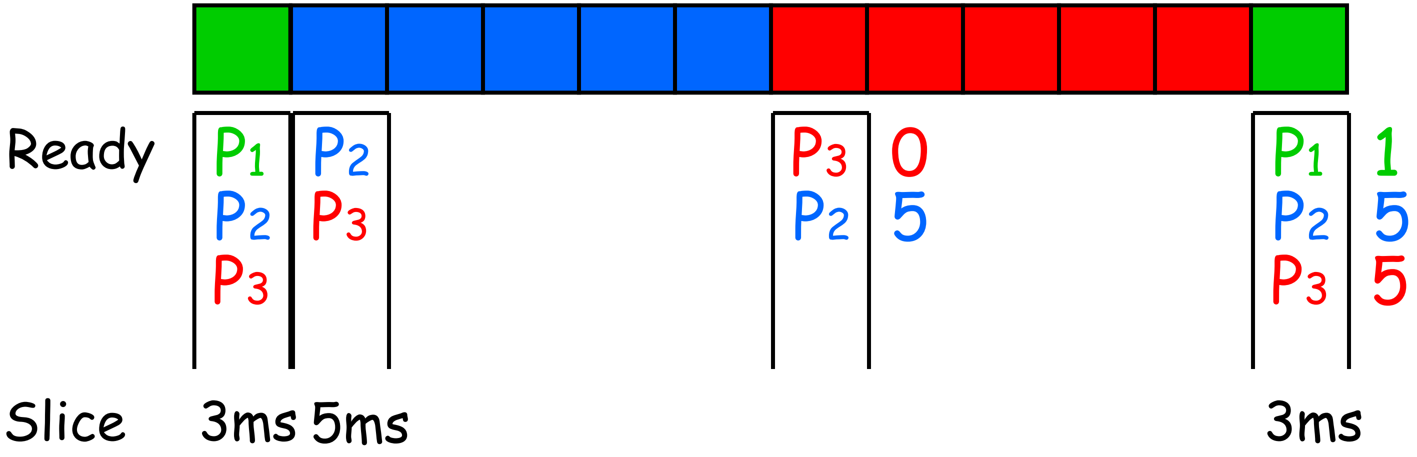
Completely Fair Scheduler (CFS)

- Approximate fair scheduling
  - Run each thread once per schedule latency (SL)
  - Weighted time slice: SL * Wi / (Sum of all Wi)

- Too many threads?
  - Lower bound on smallest time slice
  - Schedule latency = lower bound * (# threads)

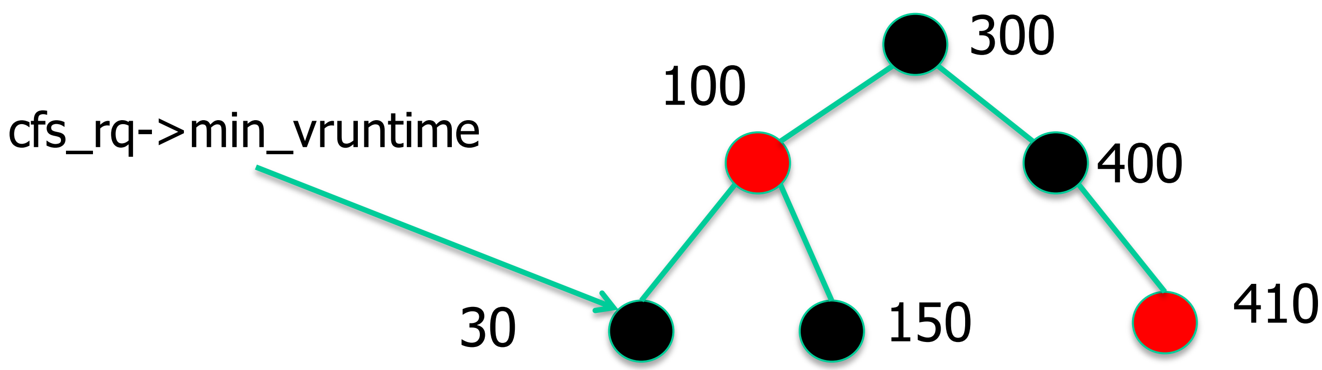
Picking the next process

- Pick proc with minimum virtual runtime so far
  - Virtual runtime: task->vruntime += executed time / Wi
- Example
  - P1: 1 ms burst per 10 ms (schedule latency)
  - P2 and P3 are CPU-bound
  - All processes have the same weight (1)



Finding proc with minimum runtime fast

- Red-black tree
  - Balanced binary search tree
  - Ordered by vruntime as key
  - O(lgN) insertion, deletion, update, O(1): find min



- Tasks move from left of tree to the right
- min_vruntime caches smallest value
- Update vruntime and min_vruntime
  - When task is added or removed
  - On every timer tick

Notable implementation details

- Integer table of nice-level to weight
  - static const int prio_to_weight[40] (kernel/sched/sched.h)
  - Nice level changes by 1 ➔ 10% weight
- cgroup
  - Fairness between users & apps, rather than threads
  - cgroup's vruntime == sum of its threads' vruntimes
- Upper bound on vruntime difference
  - New thread gets max vruntime in the RQ
  - When thread wakes up, its vruntime >= min_vruntime
- Load balancing based on many factors