CSCI 3753
Operating Systems

CPU Scheduling (Advanced)

Lecture Notes By

Shivakant Mishra

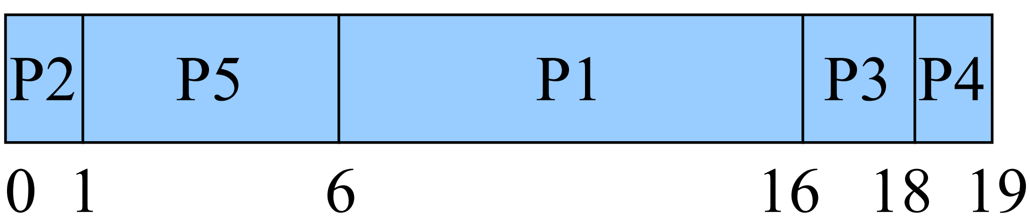
Computer Science, CU-Boulder

Last Update: 10/03/17

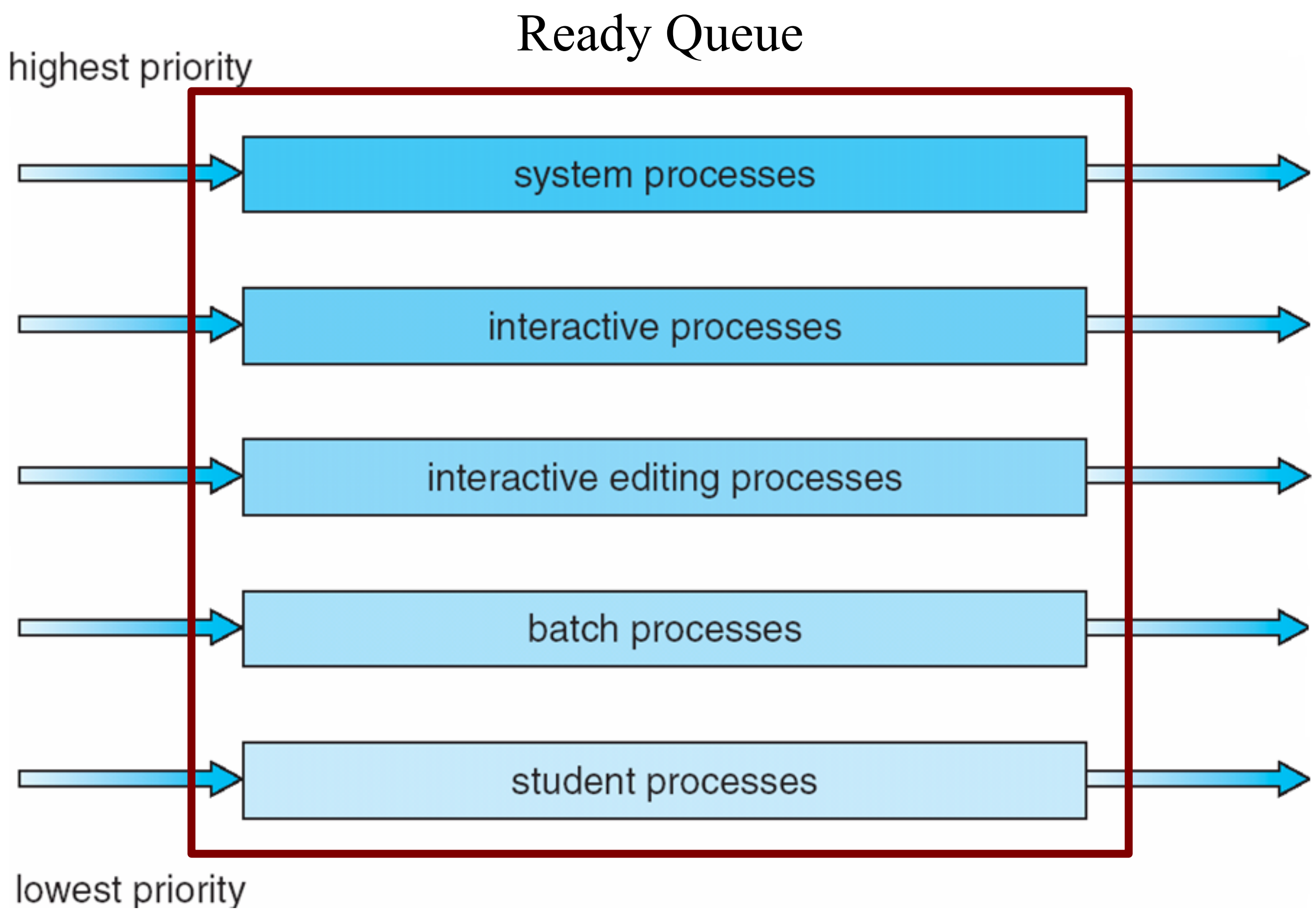
Priority Scheduling

- Assign each task a priority, and schedule higher priority tasks first
- Priority can be based on
  - any measurable characteristics of the process, or
  - some external criteria
- Can be preemptive

Process	CPU Execution Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2



Multilevel Queue Scheduling



Multi-level Queue Scheduling

- Use priorities to partition the ready queue into several separate queues
  - Different processes have different needs, e.g. foreground and background
  - If there are multiple processes with the same priority queue, need to apply a scheduling policy within that priority level
  - Don’t have to apply the same scheduling policy to every priority level, e.g. foreground gets EDF, background gets RR or FCFS
- Queues can be organized by priority, or each given a percentage of CPU, or a hybrid combination

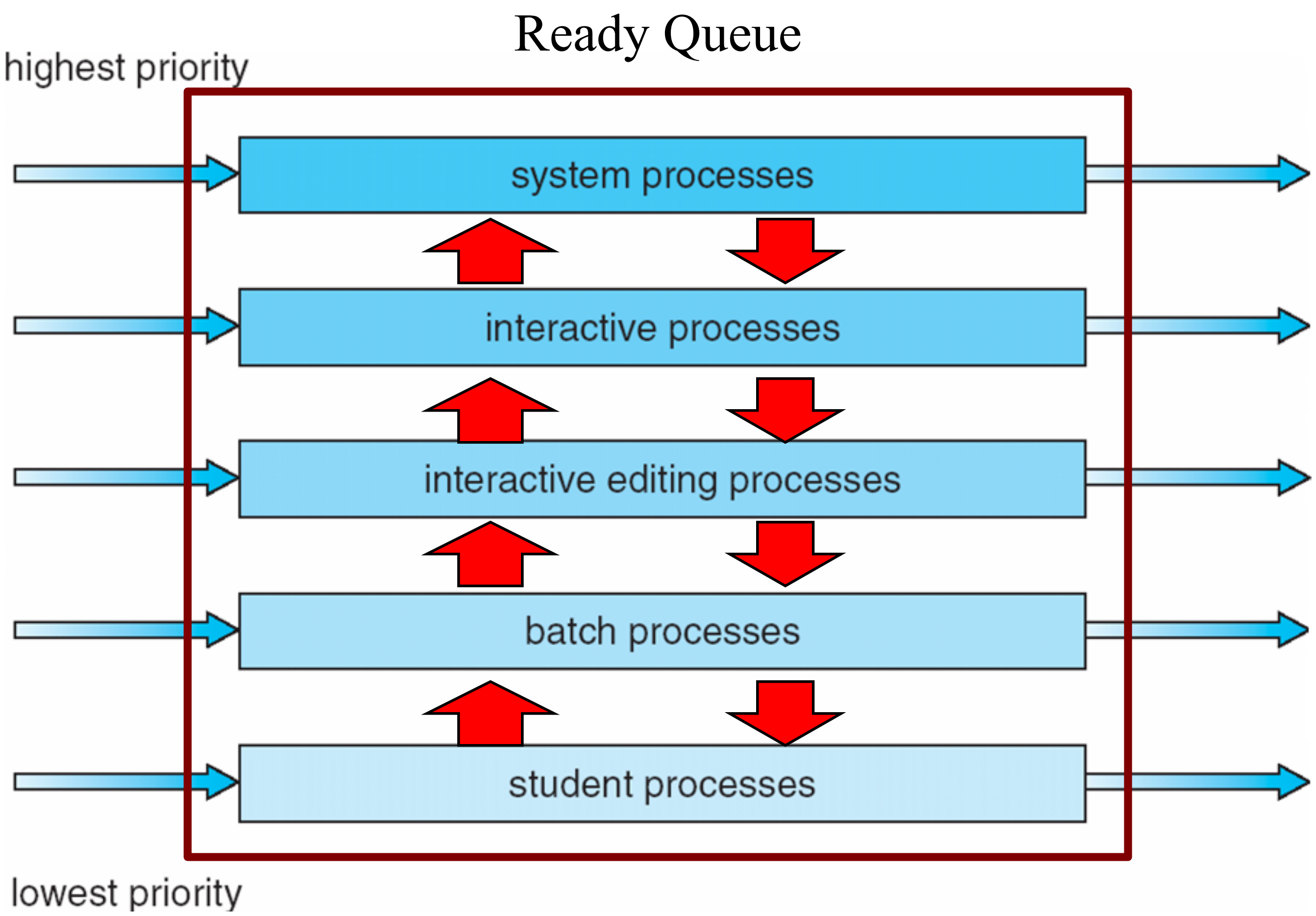
Priority Scheduling

- Preemptive priorities can starve low priority processes
  - A higher priority process always gets served ahead of a lower priority process, which never sees the CPU
- The solution is *multi-level feedback queues* that allow a process to move up/down in priority
  - Avoids starvation

Multilevel Feedback Queue

- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Multilevel Feedback Queue Scheduling



Criteria for Process Movement

1. Age of a process: old processes move to higher priority queues, or conversely, high priority processes are eventually demoted
   - Sample aging policy: if priorities range from 1-128, can decrease (increment) the priority by 1 every T seconds
   - Eventually, the low priority process will get scheduled on the CPU

2. Behavior of a process (CPU bound vs I/O bound)
   - Give higher priority to I/O bound processes: allows higher parallelism between CPU and I/O
   - A process typically alternates between bursts of I/O activity and CPU activity
   - Move a process down the hierarchy of queues during CPU burst, allowing interactive and I/O-bound processes to move up
   - Give a time slice to each queue, with smaller time slices higher up
   - If a process uses its time slice completely (CPU burst), it is moved down to the next lowest queue
   - Over time, a process gravitates towards the time slice that typically describes its average local CPU burst

Multi-level Feedback Queues

- In Windows XP and Linux, system & real-time processes are grouped in a range of priorities that are higher in priority than the range of priorities given to non-real-time processes
  - XP has 32 priorities. 1-15 are for normal processes, 16-31 are for real-time processes. One queue for each priority.
    - XP scheduler traverses queues from high priority to low priority until it finds a process to run
  - In Linux, priorities 0-99 are for important/real-time processes while 100-139 are for user processes. Lower values mean higher priorities.
    - Also, longer time quanta for higher priority tasks (200 ms for highest) and shorter time quanta for lower priority tasks (10 ms for lowest).

Announcements

- PA2 grading
  - Sign up for interview slot
  - Deadline to sign up is Friday, 10/13, 9 AM
- Programming assignment three is available on Moodle
- Midterm exam
  - Thursday, October 26 in class
  - Lecture sets 1 to 11
  - Chapters 1 to 6 and 13
  - A sample midterm exam (Fall 2016) is available on Moodle
- Readings: Chapters 1 – 6, 13 and 8

Recap …

- CPU scheduling: which process to run next
- Evaluation metrics: wait time, turn around time, response time, CPU utilization, throughput
- FCFS
- SJF
- EDF
- Round robin and weighted round robin
- Multilevel queue scheduling
- Multiple feedback queue scheduling

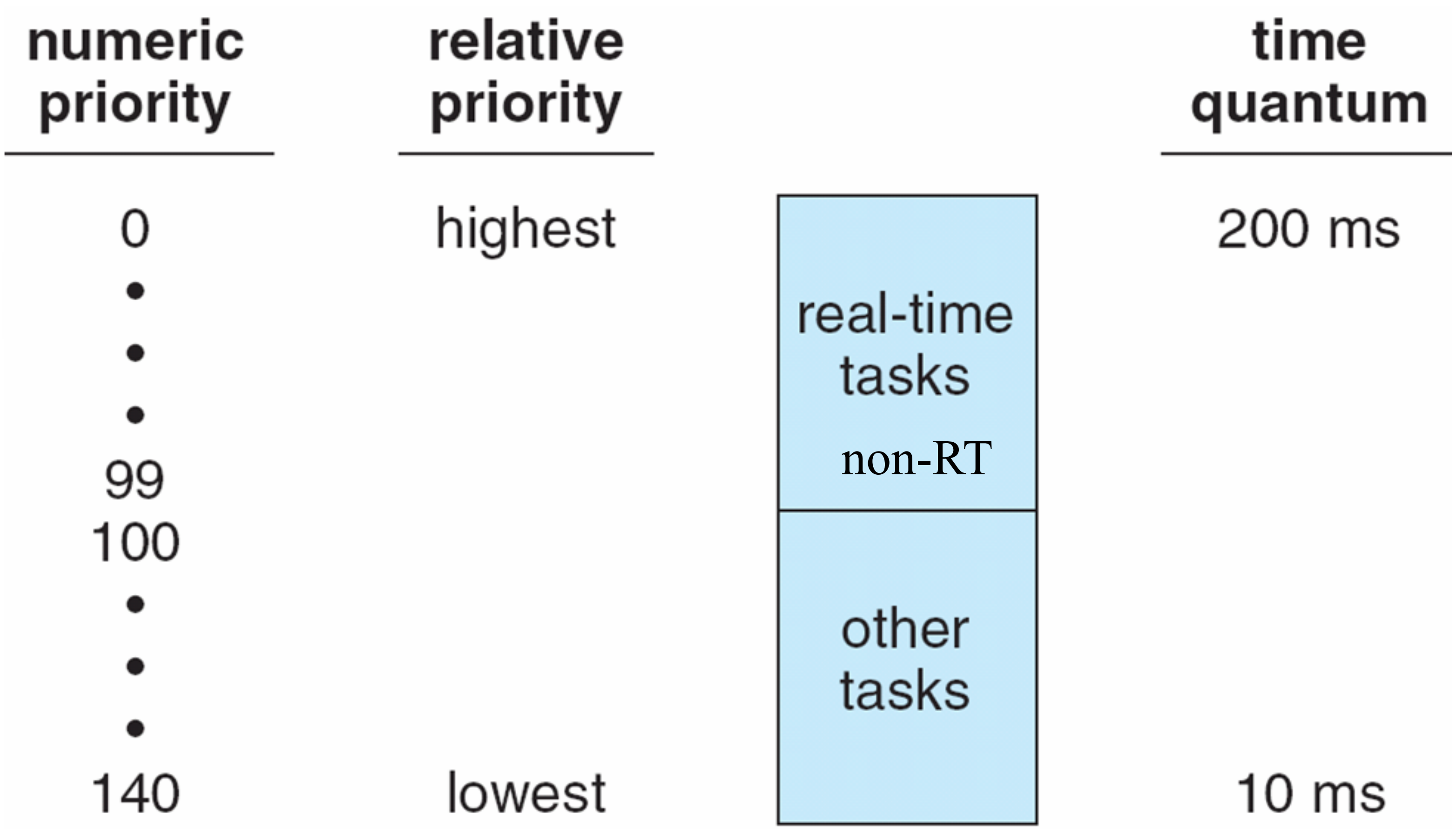
Multi-level Feedback Queues

- Most modern OSs use or have used multi-level feedback queues for priority-based preemptive scheduling
  - Windows NT/XP, Mac OS X, FreeBSD/NetBSD and Linux pre-2.6
    - Linux 1.2 used a simple round robin scheduler
    - Linux 2.2 introduced scheduling classes (priorities) for real-time and non-real-time processes and SMP support
    - Linux 2.4 introduced an O(N) scheduler – help interactive processes
    - Linux 2.6-2.6.23 uses an O(1) scheduler
    - And Linux 2.6.23+ uses a “Completely Fair Scheduler”

O(N) Scheduler

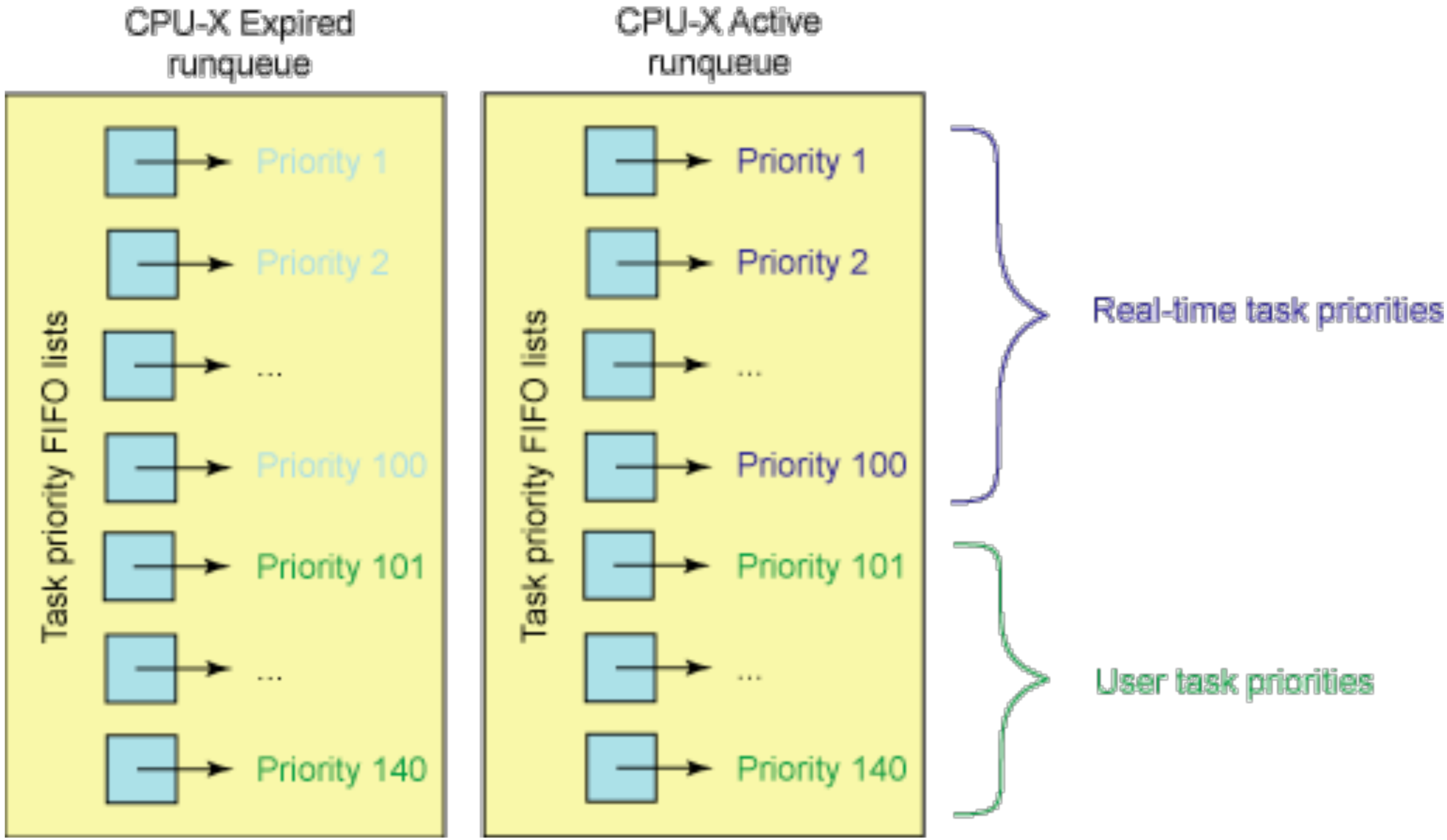
- CPU time is divided into epochs
  - Within each epoch, every process can execute up to its time slice
  - If a task does not use all of its time slice, the scheduler adds half of the remaining time slice to allow it to execute longer in the next epoch
- If an interactive process yields its time slice before it's done, then its "goodness" is rewarded with a higher priority next time it executes
- Keep a list of goodness of all tasks
  - Picking the next process to run requires iteration through all ready-to-run processes – hence O(N) – doesn't scale well

O(1) Scheduler
Priorities and Time-slice length



O(1) Scheduler in Linux

- O(1) scheduler maintains two queues: an active runqueue and an expired runqueue, each indexed by 140 priorities
- Active runqueue contains all processes with time remaining in their time slices, and expired runqueue contains all processes with expired time slices
- Once a process has exhausted its time slice, it is moved to the expired runqueue and is not eligible for execution again until all other processes have exhausted their time slices



From http://www.ibm.com/developerworks/linux/library/l-scheduler/

O(1) Scheduler

- Scheduler chooses task with highest priority from active runqueue
  - Just search linearly up the active runqueue from priority 1 until you find the first priority whose queue contains at least one unexpired task
  - # of steps to find the highest priority task is in the worst case 140, so it's bounded and depends only on the # priorities, not # of tasks - hence this is O(1) in complexity

O(1) Scheduler in Linux

- When all tasks have exhausted their time slices, the two runqueues are exchanged, and the expired runqueue becomes the active runqueue
- When a task is moved from active to expired runqueue, Linux recalculates its priority according to a heuristic
  - New priority = nice value +/- f(interactivity), where f() can change the priority by at most +/-5, and is closer to -5 if a task has been sleeping while waiting for I/O (interactive tasks tend to wait longer times for I/O, and thus their priority is boosted -5), and closer to +5 for compute-bound tasks
  - This dynamic reassignment of priorities affects only the lowest 40 priorities for non-RT/user tasks (corresponds to the nice range of +/- 20)

Completely Fair Scheduler (CFS)

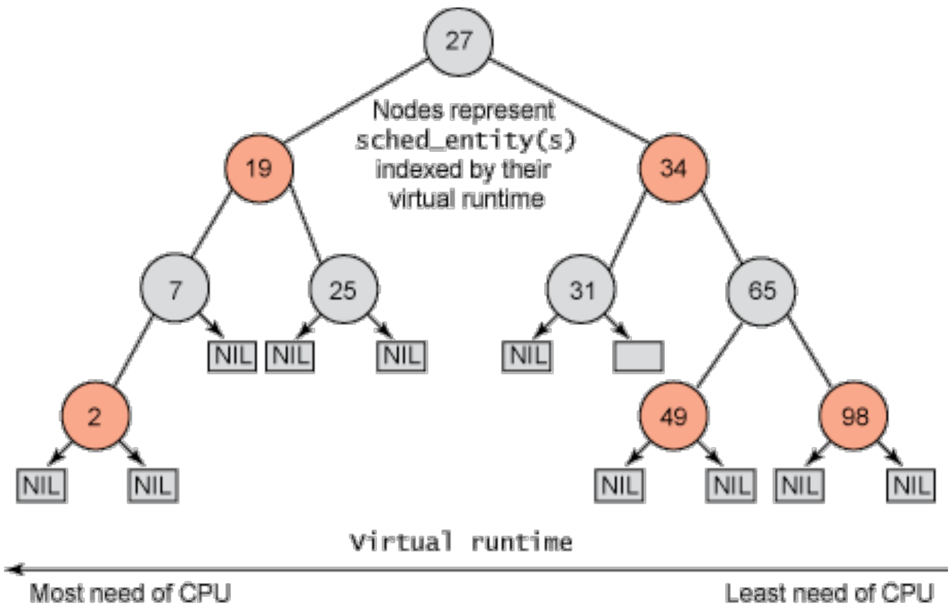
- Linux 2.6.23+ has a “completely fair” scheduler
- Heuristics of O(1) for dynamic reassignment of priorities were complicated and somewhat arbitrary
- The main idea behind the CFS is to maintain balance (fairness) in providing processor time to tasks
- Virtual run time (vruntime): The amount of time a process has used the CPU so far
  - The smaller the virtual run time, the more need for the processor. This is fair.

CFS

- Decay factor: CFS doesn't use priorities directly but instead uses them as a decay factor for the time a task is permitted to execute
  - low priority → high decay factor → the time a task is permitted to execute dissipates more quickly for a lower-priority task than for a higher-priority task
  - elegant solution to avoid maintaining run queues per priority

CFS

- Use (time-ordered) red-black tree instead of queue



- All leaves (NIL) are black
- Both children of a red node are black
- Every path from a given node to any of its descendant NIL nodes contains the same number of black nodes

From http://www.ibm.com/developerworks/linux/library/l-completely-fair-scheduler/

- Approximately balanced binary search tree
- Self balancing binary Tree: No path in the tree will ever be more than twice as long as any other
- Insert/delete/search occur in O(log *n*) time

CFS

- Lower vruntime processes are on the left side of the tree
- The scheduler picks the left-most node of the red-black tree to schedule next to maintain fairness
- The task accounts for its time with the CPU by adding its execution time to the virtual runtime and is then inserted back into the tree if runnable
- Tasks on the left side of the tree are given time to execute, and the contents of the tree migrate from the right to the left to maintain fairness

Real Time Scheduling in Linux

- Linux also includes two real-time scheduling classes:
  - Real time Round Robin
  - Real time FIFO
  - These are soft real time scheduling algorithms, not hard real time scheduling algorithms with absolute deadlines
- Only processes with the priorities 0-99