The Completely Fair Scheduler

- CFS is a significant departure from the traditional UNIX process scheduler.
- All processes are allotted a proportion of the processor's time in as "fair" of a way as possible

CFS – Nice Values

- A nice value is assigned to each task
  - Nice values range from -20 to +19
  - Lower nice value indicates a higher relative priority
  - Tasks with lower nice values receive a higher proportion of CPU processing time than tasks with higher nice values
  - The default nice value is 0

CFS – Virtual Runtime

- The virtual run time (vruntime) of a task is the actual runtime weighted by its niceness and is measured in nanoseconds
- The virtual run time is associated with a decay factor based on the task's nice value
  - nice=0, factor=1: vruntime is the same as real run time spent by task
  - nice<0,factor<1: vruntime is less than real run time spent; vruntime grows slower than real run time used
  - nice>0,factor>1: vruntime is more than real run time spent; vruntime grows faster than real run time used

CFS – Virtual Runtime

- Example: Assume a process runs for 200 milliseconds
  - Nice value is 0: vruntime will be 200 milliseconds
  - Nice value < 0 : vruntime will be less than 200 milliseconds
  - Nice value > 0 : vruntime will be greater than 200 milliseconds

- CFS selects the process with the minimum virtual runtime

CFS – Calculating vruntime

- vruntime for process with nice value i is calculated as follows:
  - Compute the time spent by the process in the CPU
  - Multiply by $weight_0/weight_i$ where
    - $weight_0$ is the weight of nice value 0
    - $weight_i$ is the weight of nice value i
- $weight_0/weight_i$ is the factor
- Weights of nice values are precomputed to avoid runtime overhead

CFS – Calculating vruntime

- Example weights
  - Weight is 1024 for nice value 0
  - Weight is 820 for nice value 1
  - Weight is 335 for nice value 5
  - Weight is 1277 for nice value -1
  - Weight is 3121 for nice value -5

- Example factors
  - Process with nice value of -1 : 1024/1277 = .80
  - Process with nice value of 1: 1024/820 = 1.24
  - Process with nice value of -5: 1024/3121 = .33
  - Process with nice value of 5: 1024/335 = 3.05

CFS – Virtual Runtime

- Example
  - Assume two tasks, A and B, with nice values of -1 and + 1 respectively
  - A and B are CPU bound and let's assume they have have the same CPU bursts
  - Which will be selected first?
    - A

CFS – Virtual Runtime

- Example
  - Assume two tasks, A and B, with nice values of -1 and + 1 respectively
  - A is CPU bound and B is I/O bound
  - Which will be selected first?
    - $runtime_A$ = 100 $runtime_B$ = 5 (real time values)
    - $vruntime_A$ = 100*.80 = 80
    - $vruntime_B$ = 5*1.24 = 5.124

CFS – Virtual Runtime

- Example
  - Assume two tasks, A and B, with nice values of -1 and + 1 respectively
  - A is I/O bound and B is CPU bound
  - Which will be selected first?
    - $runtime_A$ = 5; $runtime_B$ = 100 (real time values)
    - $vruntime_A$ = 5*.80 = 4
    - $vruntime_B$ = 100*1.24 = 153.76

CFS – Virtual Runtime

- Example
  - Assume two tasks, A and B, with nice values of -5 and + 5 respectively
  - Both A and B are CPU bound
  - Which will be selected first?
    - $runtime_A$ = 100 $runtime_B$ = 100 (real time values)
    - $vruntime_A$ = 100*.33 = 33
    - $vruntime_B$ = 100*3.05 = 305

CFS – Virtual Runtime

- Example
  - Assume two tasks, A and B, with nice values of -5 and + 5 respectively
  - A is CPU bound and B is I/O bound
  - Which will be selected first?
    - $runtime_A$ = 100 $runtime_B$ = 5 (real time values)
    - $vruntime_A$ = 100*.33 = 33
    - $vruntime_B$ = 5*3.05 = 15.25

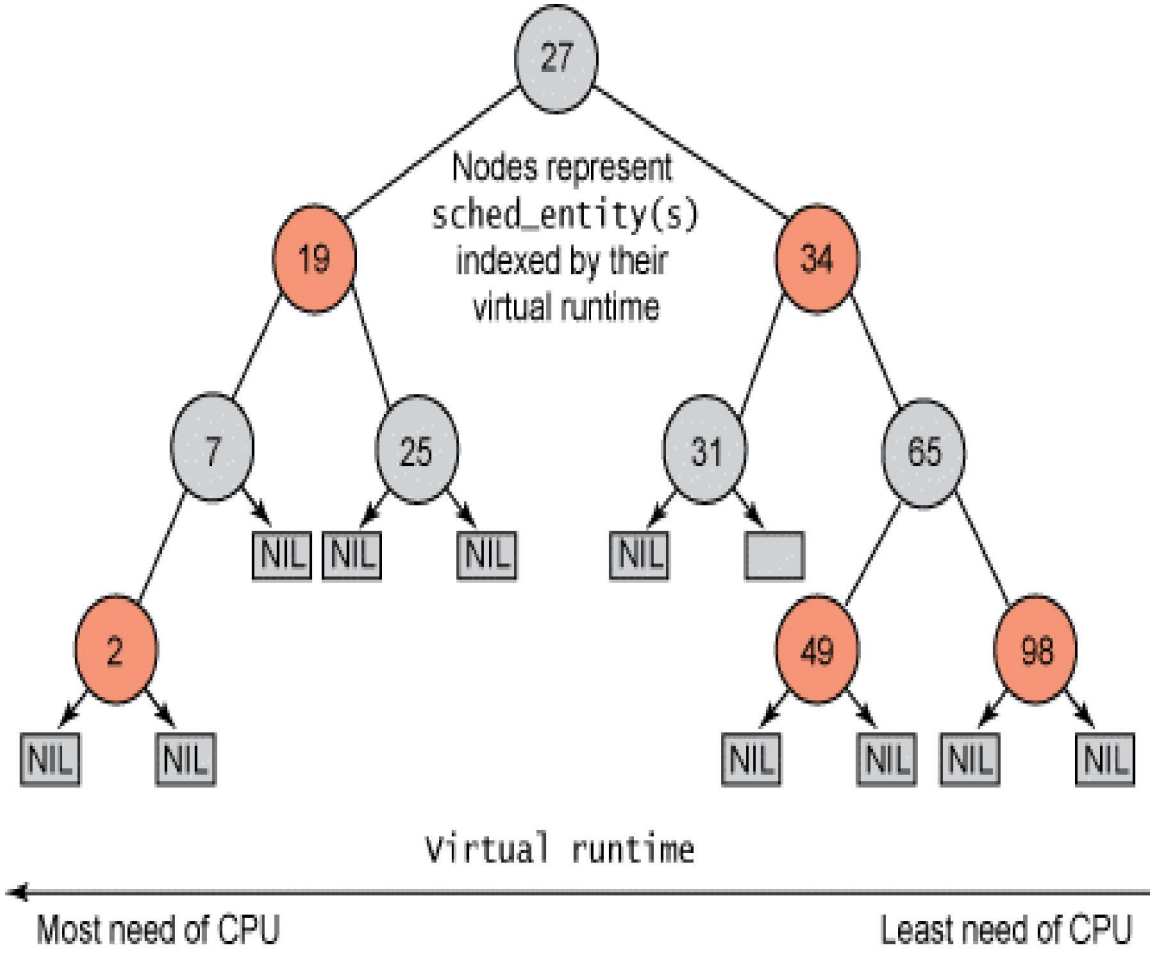
CFS - Process Selection

- CFS selects the process with the minimum virtual runtime
- Avoids having run queues per priority level
- What about a data structure that represents the collection of tasks?
  - A single queue would be slow
  - Multiple queues make sense if there are relatively small number of them
  - There are many values of virtual runtime

CFS - RB Trees

- A red-black (RB) tree is a binary search tree, which means that for each node, the left subtree only contains keys less than the node's key, and the right subtree contains keys greater than or equal to it.
- A red-black tree has further restrictions which guarantee that the longest root-leaf path is at most twice as long as the shortest root-leaf path. This bound on the height makes RB Trees more efficient than normal BSTs.
- Operations are in O(log n) time.

CFS -- Task Selection



- The key for each node is the vruntime of the corresponding task.
- To pick the next task to run, simply take the leftmost node.
  - This node is pointed to by a variable to reduce traversals

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

CFS- "Time Slice" Calculation

- CFS identifies a targeted latency (TL) which is an interval of time during which every runnable task should run at least once
- Proportions of CPU time are allocated from the value of targeted latency
- Targeted latency can increase if the number of active tasks grows beyond a threshold value

CFS- "Time Slice" Calculation

- Proportions of CPU time is calculated based on a mapping of a priority (nice value) to a weight value
  - Examples: -20 is mapped to 88761, 0 is mapped to 1024
- The general idea is that
  - Every process that changes priority up by one level gets 10% less CPU power
  - Every process that changes priority down by one gets 10% more CPU power

CFS- "Time Slice" Calculation

- The mapping from priorities to weights uses an array
- The array contains one entry for each nice level
- The multiplier between the entries is 1.25

CFS "Time Slice" Calculation

- Example
  - Targeted latency = 20 ms
  - Two tasks, t_1 and t_2, with nice values of 0
  - The weight value for nice 0 is 1,024
  - The share for each task is 1024/(1024+1024) = 0.5 and thus each task will have a "time slice" of 10 ms

CFS "Time Slice" Calculation

- Example
  - Targeted latency = 20 ms
  - Two tasks, t_1 and t_2, with nice values of 0 and 5 respectively
  - The weight value for nice 0 is 1,024 and the weight for nice 5 is 335
  - The share for each task is 1024/(1024+335) = 0.75 and thus
    - task t_1 will have a "time slice" of 0.75*20 = 15 ms
    - task t_2 will have a "time slice" of 5 ms

CFS

- No static slices
  - The switching rate depends on the system load
- Each process receives a proportion of the processor's time
  - Length depends on how many other processes are running