# Linux CFS

Yajin Zhou (http://yajin.org)

Zhejiang University

credits: Dr. C.C. Lee, Abdulrahman Idlbi

# CPU affinity

- CPU affinity: less overhead, in cache
- Soft affinity means that processes do not frequently migrate (under load balancing)  between processors
- Hard affinity means that processes run on processors you specify

Reason 1: You have a hunch – computations

Reason 2: Testing complex applications – linear scalability?

Reason 3: Running time-sensitive, deterministic processes

*http://www.ibm.com/developerworks/linux/library/l-affinity/index.html*

** sched_setaffinity (…)  set CPU affinity mask*

# Run Queue Per Processor

- Eliminates the need to serialize one global runqueue and eliminates waiting for the scheduler

  *Multiple processors can now execute the scheduler*

  *concurrently, each on their own local queue.*

- One runqueue per *CPU* naturally **improved *CPU* affinity**
  - Because runqueues are local to each processor, an interrupted task will resume on the same processor it was running on before the interruption, and any performance improvement realized by local processor and memory caches is maintained.
  - To avoid the problem of imbalance, the scheduler will, when appropriate, migrate processes between CPUs to balance the workload. (under soft CPU affinity)

# Process (Task) Basics

- Process States
  - TASK_RUNNING (run or ready)
  - TASK_INTERRUPTIBLE (sleeping or blocked, may be waken by signal)
  - TASK_UNTERRUPTIBLE (sleeping/blocked, only event can wake this task)
  - TASK_STOPPED (SIGSTOP, SIGTTIN, SIGTTOU signals)
  - TASK_ZOMBIE (pending for parent task to issue wait)

# Process (Task) Scheduling

- Preemptive
- Scheduler Classes (priority for classes)
  - Real-time: FIFO and RR (timeslice), fixed priority
  - Normal (SCHED_NORMAL)
- SMP (Run queue/structure per CPU, why?)
- Processor Affinity (Soft & Hard)
- Load balancing

# Process (Task) Scheduling

- Two process-scheduling Classes:
  - **Normal time-sharing (dynamic)**

    (Nice value: 19 to -20, with default 0 = 120)

  - Real-time algorithm (FIFO/RR) - Soft

    Absolute priorities (static): 0-99

    FIFO run till Exit , Yield, or Block

    RR run with time slice

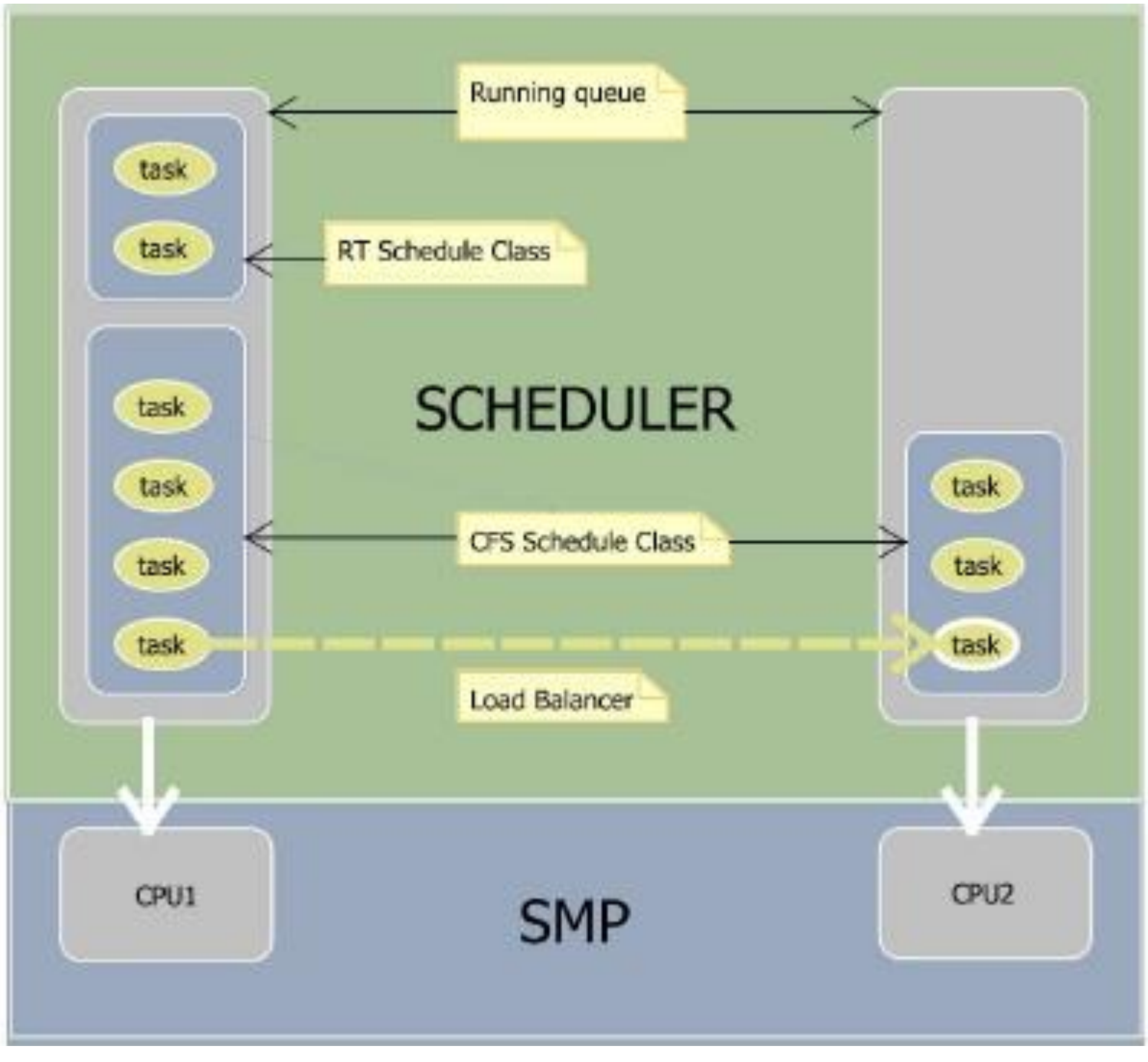
    Preemption possible with priority

- Normal Processes: **to be discussed here**

# Completely Fair Scheduler (CFS)

- Since Kernel 2.6.23
- CFS Aiming at
  - Giving each task a fair share (portion) of the processor time (Completely Fair)
  - Improving the interactive performance of scheduler for desktop.
  - Introduces simple/efficient algorithmic approach (red-black tree) with O(log N).

# Completely Fair Scheduler (CFS)



# CFS – Processor Time Allocation

- Select next task that has run the least CPU. How long a process should run is a function of the total number of runnable processes and its niceness (default: 1 ms as minimum granularity)

- Nice values are used to weight the portion of processor a process is to receive. Each process will run for a “timeslice” proportional to its weight divided by total weight of all runnable processes.

- Assume TARGETED_LANTENCY = 20ms and two threads (niceness of 0 and 5 – corresponding to relative weight 3:1 by a special algorithm). Therefore CPU allocation time would be 15ms (for niceness 0) and 5ms (for niceness 5). Here, CPU portion is determined only by the relative value difference. So if two threads with niceness 10 and 15, the result will be the same (the value difference is still 5)

# CFS – The Virtual Runtime (vruntime)

- The virtual runtime (*vruntime)* is the actual runtime (the amount of time spent running) weighted *by the number of runnable processes and the niceness as follows:*

  *nice=0, factor=1; vruntime is 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.*

  *(The virtual runtime is measured in nano seconds)*

- Every time a thread runs for *t* ns, *vruntime* += t (weighted by task niceness)
- The virtual runtime (*vruntime*) is used to account for how long a process has run. CFS will then pick up the process with the smallest *vruntime*.

# CFS – Task Selection

- CFS select the task with the minimum virtual runtime i.e. *vruntime*

- CFS use a *red-black tree (rbtree – a type of self-balancing binary search tree)* to manage the list of runnable processes and efficiently (algorithm) find the process with the smallest *vruntime*

- The selected task with the smallest *vruntime* is the leftmost node in the (*rbtree)* tree.

# CFS – Task just Created or Awaken

- A new task is created

vruntime = Current min_vruntime (reduced by some adjustment) and it will be inserted into the *rbtree*

- A task is awakened from blocking

vruntime = Maximum (old vruntime, min_vruntime - targeted_latency).

The currently running task will be preempted and the awakened task will be scheduled. Using “min_vruntime – Target_Latency” as a lower bound on an awakened task prevents a task that blocked for a long time from monopolizing the CPU

# CFS – Group Scheduling

- In plain CFS, if there are 25 runnable processes, CFS will allocate 4% to each (assume same). If 20 belong to user A, and 5 belong to user B, then user B is at an inherent disadvantage.

- Group scheduling will first try to be fair to the group and then individual in the group, i.e. 50% to user A and 50% to user B.

- Thus for A, the allocated 50% of A will be divided fairly among A's 20 tasks. For B, the allocated 50% will be divided fairly among B's 5 tasks.

# CFS – Run Queue (Red-Black Tree)

- Tasks are maintained in a time-ordered (i.e. vruntime) red-black tree for each CPU
- Red-Black Tree: Self-balancing binary search tree
  - Balancing is preserved by painting each node with one of two colors in a way to satisfy certain properties. When the tree is modified , the new tree is rearranged and repainted to restore the coloring properties.
  - The balancing of the tree can guarantee that no leaf can be more than twice as deep as others and the tree operations (searching/insertion/deletion/recoloring) can be performed in O(log N) time
- CFS will switch to the leftmost task in the tree, that is, the one with the lowest virtual runtime (most need for CPU) to maintain fairness.

# CFS – Run Queue (Red-Black Tree)