The Rotating Staircase Deadline Scheduler

Jonathan Corbet
LWN.NET

March, 2007

CPU scheduling seems to be one of those eternally unfinished jobs. Developers can work on the CPU scheduler for a while and make it work better, but there will always be workloads which are not served as well as users would like. Users of interactive systems, in particular, tend to be sensitive to scheduler latencies. In response, the current scheduler has grown an elaborate array of heuristics which attempt to detect which processes are truly interactive and give them priority in the CPU. The result is complicated code - and people still complain about interactive response.

Enter Con Kolivas, who has been working on improving interactivity for some time. His latest proposal is the Rotating Staircase Deadline Scheduler (RSDL), which attempts to provide good interactive response with a relatively simple design, complete fairness, and bounded latency. This work takes ideas from Con's earlier staircase scheduler (covered here in June, 2004), but with a significantly different approach.

Like many schedulers, the RSDL maintains a priority array, as is crudely diagrammed at Figure (1). At each level there is a list of processes currently wanting to run at that priority; each process has a quota of time it is allowed to execute at that priority. The processes at the highest priority are given time slices, and the scheduler rotates through them using a typical round- robin algorithm.

When a process uses its quota at a given priority level, it is dropped down to the next priority and given a new quota. That process can thus continue to run, but only after the higher-priority processes have had their turn. As processes move down the staircase, they increasingly must contend with the lower- priority processes which have been patiently waiting on the lower levels. The end result is that even the lowest-priority processes get at least a little CPU time eventually.

An interesting feature of this scheduler is that each priority level has a quota of its own. Once the highest priority level has used its quota, all processes running at that level are pushed down to the next-lower level, regardless of whether they have consumed their individual CPU time quotas or not. As a result of this "minor rotation" mechanism, processes waiting at lower priority levels need only cool their

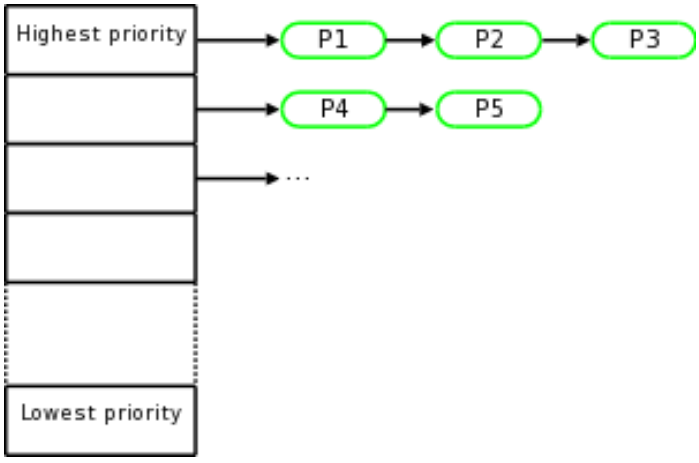

Figure 1: RSDL Priority Queues

heels for a bounded period of time before all other processes are running at their level. The maximum latency for any process waiting to run is thus bounded, and can be calculated; there is no starvation with this scheduler.

As processes use up their time, they are moved to a second array, called the "expired" array; there they are placed back at their original priority. Processes in the expired array do not run; they are left out in the cold until no more processes remain in the currently active array - or until all processes are pushed off the bottom of the active array as a result of minor rotations. At that point, a "major rotation" happens: the active and expired arrays are switched and the whole series of events restarts from the beginning.

The current scheduler tries to locate interactive tasks by tracking how often each process sleeps; those seen to be interactive are then rewarded with a priority boost. The RSDL does away with all that. Instead, processes which sleep simply do not use all of their time at the higher priority levels. When they run, they are naturally advantaged over their CPU-hungry competition. If a process sleeps through a major rotation, its quota goes back into the run queue's priority-specific quota value. Thus, it will be able to run at high priority even if other high-priority processes, which have been running during this time, have been pushed to lower priorities through minor rotations. All of this should add up to quick response from interactive applications.

A few benchmarks posted by Con show that systems running with RSDL perform slightly better than with the stock 2.6.20 scheduler. The initial reports from testers have been positive, with one person urging that RSDL go into 2.6.21. That will not happen at this point in the release cycle, but Linus is favorable to including RSDL in a future kernel:

> I agree, partly because it's obviously been getting rave reviews so far, but mainly because it looks like you can think about behaviour a lot better, something that was always very hard with the interactivity boosters with process state history.

Con has recently been heard to complain about difficulties getting his interactivity improvements into the mainline. This time around, however, he may find the course of events to be rather more gratifying.