Work-conserving scheduler

From Wikipedia, the free encyclopedia

In computing and communication systems, a **work-conserving scheduler** is a scheduler that always tries to keep the scheduled resource(s) busy, if there are submitted jobs ready to be scheduled. In contrast, a **non-work conserving scheduler** is a scheduler that, in some cases, may leave the scheduled resource(s) idle despite the presence of jobs ready to be scheduled.

For example, when dealing with networking and packet scheduling, a work-conserving scheduler[1][2] leaves the channel idle only when there are no packets to transmit, whereas a non-work conserving one might leave the channel idle also with packets pending to be transmitted.

Similarly, when referring to CPU scheduling, i.e., threads or processes scheduled over one or more available processors or cores, a work-conserving scheduler[3] ensures that processors/cores are not idle if there are processes/threads ready for execution.

Non-work conserving schedulers are sometimes useful to enhance predictability and reduce termination jitter for the activities carried out by a computing and communication system. In multi-processor systems they're useful to enhance performance in some scenarios.[4] [5] Sometimes, a non-work conserving scheduler may be useful to enhance stability of a system; indeed, one of the issues with the real-time scheduling disciplines `SCHED_RR` and `SCHED_FIFO` as standardized by POSIX,[6] was that, while developing and testing a software making use of these disciplines, a bug causing an infinite loop would entirely freeze a single-processor machine, as no non-real time process would have any chance to be scheduled ever. In order to prevent this to happen, in the Linux kernel the *throttling* mechanism was added that didn't allow real-time tasks to run for longer than 950ms every second (by default), leaving a minimum of CPU resources to the background OS tasks and other processes like a terminal, so that an operator would still be able to kill the offending real-time process and keep control of the machine if needed.

References

1. [1] (http://www.csee.umbc.edu/~pmundur/courses/CMSC691C/scheduling.ppt) Padma Mundur, Improving QOS in IP Networks (course material for Multimedia Networking (http://www.csee.umbc.edu/~pmundur/courses/CMSC691C/))
2. [2] (http://www.cl.cam.ac.uk/teaching/0405/DigiComm2/sched.ppt) Jon Crowcroft, Scheduling and queue management (course material for Digital Communications II (http://www.cl.cam.ac.uk/teaching/0405/DigiComm2/))
3. [3] (http://www.springer.com/in/book/9780387237015) G. Buttazzo, G. Lipari, L. Abeni, M. Caccamo, Soft Real-Time Systems: Predictability vs. Efficiency, Springer 2005
4. [4] (http://www.bioperf.org/FSS06.pdf) A. Fedorova, M. Seltzer and M.D. Smith, "A non-work-conserving operating system scheduler for SMT processors," in Proc. USENIX 2005 Annual Technical Conference, Anaheim, California, April 2005
5. [5] (http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4625838) J. C. Sáez, J. I. Gomez and M. Prieto, "Improving Priority Enforcement via Non-Work-Conserving Scheduling," Parallel Processing, 2008. ICPP '08. 37th International Conference on, Portland, OR, 2008, pp. 99-106.
6. [6] (http://www.opengroup.org/onlinepubs/009695399) IEEE Standard for Information Technology – Portable Operating System Interface, POSIX.1b, Real-time extensions (IEEE Std 1003.1b-1993)