Wai Phyo | CS-431 | EX-4

Completely Fair Scheduler (CFS) Report

Background

CFS was invented by Ingo Molnár for Linux Systems. It replaced O(1) scheduler starting from Linux 2.6.23. CFS is claimed to be completely fair to both I/O bound and CPU bound processes which are in the ready state. There are a few other schedulers which came out after CFS. For example, BFS, Brain Fuck Scheduler, is used as an alternate to CFS in some Desktop systems. BFS focuses on responsiveness of each process while CFS focuses on fairness.

Goals

The general goal of CFS is the same as other schedulers. It is to be fair, balanced, and enforces the respective policies. The main goal of CFS is to create a process scheduler which has a simple core algorithm, and the algorithm has to be completely fair in selecting all ready-state processes. Theoretically, a completely fair algorithm should allow all processes to share the CPU equally. For example, if there are ‘n’ numbers of processes in the ready-state queue, each process should get 1/n unit time from CPU.

Implementation

Practically, it is impossible to split CPU to ‘n’ number of tiny slots for each process to use as CPU can only run one process at a time. Hence, there is a workaround for this practical scenario. This is the main scheduling algorithm for CFS.

How it Makes Scheduling Decision

CFS uses virtual runtime of each process for the scheduling algorithm. It is a new data field that is attached to the process structure inside the kernel. It contains the amount of time it has run on each CPU. For a completely new process, the number starts with 0 in theory. Usually, all new processes are run briefly to get some number. Whenever the process runs, the new quantum time (the time it runs) is added to the virtual runtime. When CPU chooses which process to run next, it chooses the process with the smallest number of virtual runtime. In addition, CFS also allows priorities for each process in the form of “nice time”. The higher the priority, the smaller the “nice time” number is. When virtual runtime for each process is updated, CFS multiplies the quantum time it executed with the “nice time” number. The idea behind this calculation is to have smaller virtual runtime for processes with higher priorities. Hence, they have higher chance to have the smallest virtual runtime, and to be selected again.

When it Makes Scheduling Decision

CFS makes scheduling decision at the beginning of each epoch by deciding how many processes are in the ready state. CFS’ decision is based on the duration of epoch of the CPU. Epoch is a fixed time duration for CPU. Epoch time can be different for different systems such as 4ms, 2ms, and so forth. Each epoch should be equally divided for all ready-state processes. For epoch with ‘x’ seconds, each process is allowed ‘x / n’ seconds. At the beginning of each epoch, CFS will choose process with the lowest virtual run time. The process will be run for ‘x / n’ seconds, and its virtual runtime will be updated for ‘(x / n) * nice_time’. Updating virtual runtime will ensure it does not have the smallest virtual runtime which allows CFS to choose other processes next time.

Data Structure

CFS chooses red-black tree as its main data structure. It deviates from previous schedulers which uses queues, lists, or heaps as their data structures. The main idea behind choosing red-black tree is that processes will be jumping around in the data structure because of CFS algorithm. Hence, efficiency is really important for insertion and removal of processes. The best complexity that it can get is from binary search trees. Red-black tree is a better version of binary search tree which is always balanced, hence ensuring O($\log_2$ n) complexity.

Time Complexity

CFS' time complexity is O($\log_2$ n) because of red-black tree. Due to CFS's algorithm, all ready-state processes are removed, and inserted again at each epoch. When they are inserted, processes can be inserted anywhere in the data structure depends on their virtual runtime. Since they are inserted constantly, they choose binary search tree for O($\log_2$ n) complexity. However, binary search tree can become out of balanced, and in worst case, it will become a linked list which is O(n). Hence, CFS chooses a self-balancing red-black tree which will ensure it is O($\log_2$ n) at all times. For better performance, CFS has a pointer which will point to the left most leaf child in the red-black tree. It is the process with the minimum virtual runtime. This allows CFS to have O(1) when selecting which process to run next In general, CFS has a complexity of O(log n).

Ubuntu Settings to Update Scheduler Parameter

The Linux utility to update the schedulers' settings is "schedtool". It can be installed any Linux systems. For Ubuntu, it can be installed using "*sudo apt install schedtool*". It can view the policies for each process, change the scheduling algorithm for all processes by appending with options such as -N for normal (CFS), -F for First-In-First-Out, -R for Round Robin, -B for Batch, and so forth. The full command is "*schedtool -<scheduler option>*". Some schedulers require priority number. It can also update the priority level of processes for CFS by changing the nice levels with the option "-n <new number>". Since FIFO and Round-Robin uses a different priority system, the tool has a different option to update this priority as well with the option "-prio". One final useful command is the affinity command which is useful when there are many CPUs, and wanted to dedicate 1 CPU to only selected process. The affinity number is constructed as hexadecimal, and it will run on CPUs which index is set as 1. 0x1, 0001_2 will run on CPU=0. 0x2, 0010_2 will run on CPU-1. 0x3, 0011_2 will on both CPU-0 and CPU-1. An example of this command is "*schedtool -a<affinity number in hex> <pid>*".

References

[1]"Ubuntu Manpage: schedtool - query and set CPU scheduling parameters", Manpages.ubuntu.com, 2017. [Online]. Available: http://manpages.ubuntu.com/manpages/precise/man8/schedtool.8.html. [Accessed: 05- Feb- 2017].

[2]"W5 L5 Completely Fair Scheduling", YouTube, 2017. [Online]. Available: https://www.youtube.com/watch?v=MkJfuI5_hjc. [Accessed: 04- Feb- 2017].

[3]"Inside the Linux 2.6 Completely Fair Scheduler", Ibm.com, 2017. [Online]. Available: https://www.ibm.com/developerworks/library/l-completely-fair-scheduler/. [Accessed: 04- Feb- 2017].

[4]"Cite a Website - Cite This For Me", Kernel.org, 2017. [Online]. Available: https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt. [Accessed: 04- Feb- 2017].

[5]V. Seeker, "Process Scheduling in Linux", semanticscholar.org, 2013. [Online]. Available: https://pdfs.semanticscholar.org/e541/4009ebae07114a4d290d99978a63de364a97.pdf. [Accessed: 04- Feb- 2017].