9/12/2022

1DV512 Operating system

Individual

Assignment 2

Yetnayet Edeglign Belachew
LINNAEUS UNIVERSITY

1. Does ULE support threads, or does it support processes only? How about CFS?

Yes, both ULE and CFS are both designed to schedule large numbers of **threads** on large multicore machines.
ULE keeps track of the number of threads in the key with each nice value. It also keeps the current minimum nice value, that is, the least nice process.

However, **CFS** treats the KVM threads as normal tasks without considering their unique features such as thread allocation mechanism and lock inside a guest virtual machine, which may harm the KVM virtualization performance. **ULE & CFS support threads**.

2. How does CFS select the next task to be executed?

To pick up the next process, the scheduler selects the task that has minimal runtime. Namely, the process that runs the least.
A process accumulates runtime only while it is running. So, by picking the task that has the minimal runtime we pick the task that runs the least. On the other hand, a process that waits a lot - does not accumulate runtime. Therefore, its runtime is low. And if its wait time is the longest- its runtime will be the lowest - and it will be picked to run next [1].

3. What is a *cgroup* and how is it used by CFS? Does ULE support *cgroups*?

Threads of the same application are grouped into a structure called a **cgroup**. A **cgroup** has a runtime that corresponds to the sum of the runtimes of all of its threads. CFS then applies its algorithm on cgroups, ensuring fairness between groups of threads. When a cgroup is chosen to be scheduled, its thread with the lowest runtime is executed, ensuring fairness within a cgroup. Cgroups can also be nested. For instance, the system automatically configures cgroups to ensure fairness between different users, and then fair-ness between the applications of a given user.A **cgroup** is a collection of processes that are bound to a set of limits or parameters defined via the cgroup filesystem.

No, ULE does not support cgroups, but rather considers each thread as an inde-pendent entity.

4. How many queues in total does ULE use? What is the purpose of each queue?

ULE uses two-run queues to schedule threads: one run queue contains interactive threads, and the other contains batch threads. A third run queue called idle is used when a core is idle. This run queue only Continis the idle task.

The goal of having two-run queues is to give priority to interactive threads. Batch processes usually execute without user interaction, and thus scheduling latency is less important. ULE keeps track of the interactivity of a thread using an interactivity penalty metric between 0 and 100. This metric is defined as a function of the timer a thread has spent running and the times a thread has spent voluntarily sleeping (not including the time spent waiting for the CPU) and is computed [2].

5. How does ULE compute priority for various tasks?

ULE uses the interactivity score to derive the **priority**. After this, the nice value **is** added to the **priority**, which may actually lower the **priority** in the case of negative nice values. This generally gives interactive **tasks** a chance to run sooner than non-interactive **tasks** when they **are** placed on the same queue [3].

6. Do CFS and ULE support task preemption? Are there any limitations?

CFS : Full preemption is enabled, so yes, sometimes. FreeBSD ULE : No full preemption by default. Only kernel threads can preempt others.

The performance difference between ULE and CFS is explained by different choices regarding thread preemption. In **ULE**, full **preemption is disabled**, while CFS preempts the running thread when the thread that has just been woken up has a vruntime that is much smaller than the vruntime of the currently executing thread (1ms difference in practice). In CFS, ab is preempted 2 million times during the benchmark, while it never preempted with ULE [3].

7. Did Bouron et al. discover large differences in per-core scheduling performance between CFS and ULE? Which definition of "performance" did they use in their benchmark, and why?

No, the average performance difference is 1.5%. The main difference between CFS and ULE in per-core scheduling is in the handling of batch threads: CFS tries to be fair to all threads, while ULE gives priority to interactive threads. And Bouron also mentioned in the "evaluation for pre core scheduling "that the main difference between CFS and ULE in pre-core scheduling is in the handling of Bach threads. The CFS is designed to be fair in all threads but ULE prioritize only to interactive threads. Subsequently,

- Which definition of "performance" did they use in their benchmark, and why?

The performance definition for database workloads and NAS application, they compare the number of operations per/second used and other application they compare "1/execution time". Bouron also sated that "we show that they behave differently even on simple workloads, and that no scheduler performs better than the other on all workloads." the higher the "performance", the better a scheduler performs, the performance difference between CFS and ULE on a single core, the application executes faster with ULE than CFS.

- **The performance of CFS and ULE** used both synthetic benchmarks and realistic applications. Fibo is a synthetic application computing Fibonacci numbers.

8. What is the difference between the multi-core load balancing strategies used by CFS and ULE? Is any of them faster? Does any of them typically reach perfect load balancing?

The difference of load balancing strategy used by CFS is well suited for solving large imbalance loads in the system, it is less suited when a perfect load balance is important for performance.
In a multicore setting, Linux's CFS evens out the amount of work on all cores of the machine. This is different from an evening out the number of threads. For instance, if a user runs 1 CPU- intensive thread and 10 threads that mostly sleep, CFS might schedule the 10 mostly sleeping threads on a single core.

The load balancing strategy in the ULE is more preferable in the MG, from the NAS benchmark than in the CFS.MG, from the NAS benchmark suite, benefits the most from ULE's load balancing strategy: it is 73% faster on in the ULE than on the CFS.MG contains many threads in the cores in the machine.

• **CFS** converges **faster** towards a balanced load, but **ULE achieves a better load balance** in the long run. The performance difference between CFS and ULE on a single core, the application executes faster with ULE than CFS.

[1] https://stackoverflow.com/questions/28312454/linux-cfs-how-to-select-the-next-process

[2] **The Battle of the Schedulers: FreeBSD ULE vs. Linux CFS** Justinien Bouron, Baptiste Lepers, Sébastien Chevalley, Willy Zwaenepoel EPFL Redha Gouicem, Julia Lawall, Gilles Muller, Julien Sopena Sorbonne University, Inria, LIP6
https://www.usenix.org/sites/default/files/conference/protected-files/atc18_slides_bouron.pdf

[3] ULE: A Modern Scheduler for FreeBSD

Jeff Roberson
The FreeBSD Project jeff@FreeBSD.org