Linnéuniversitetet
Kalmar Växjö

Individual Assignment 2

Report and answers of all tasks of second assignment of Opration Systems 1DV512-HT21

Author: Saleh Shalabi

ss225bx@student.lnu.se

Semester: HT21

Course code: 1DV512

Date: 09/12/2021

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

Both ULE and CFS are done to “schedule large number of threads on large multicore machines”. The answer is yes both of them support threads.

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

The CFS queuing algorithm divides CPU cycles between the threads depending on their proirity that is represented by their niceness, the higher niceness the lower proierity.

The schudel in CFS is done by the *vruntime* which can be culculated by deviding the time a thred used from CPU time by its priority. That gurantees that threads with same priority shares core resources fairly. In the schedule the thread with lowest vruntime runs next when a current running thread is preempted.

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

The *cgroup* in CFS are a structure of threads of same application grouped toghether. That means the vruntime of it is the sum of all threads runtime in it. The CFS applies the scheduling algorithm om the cgroup, which will ensure fairness between groups.

The ULE does not support the cgroup structure.

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

The ULE uses three queues. two to schdule threads. The first one contains interactive threads and the seconde that contains batch threads. The last one contains only idle task and is used when the core is idle.

Two runqueus is used to give proirity to the interactive threads. While the batch is usualy used to run with out user interaction, which make its scheduling latency less important.

5. How does ULE compute priority for various tasks?

In the ULE the for the interactive threads the priorety is a linear interpolation of their *score*, the less the score is the higher priorety it has. In the other hand the batch threads priority is deppending on their runtime, the more time a thread spend running the lower priorety it has. To get a linear effert on the priority the niceness value is added.

To classify a thread's priority the ULE computes first the *score* of the thread which is defined by *interactivity penalty* + niceness . Interactivity penalty is a matric between 0 and 100, and its defined as a function of time ***r*** a thread spend runing and time **s** which is the time a thread spend voluntarily sleeping. If its score is under a certain threshold value then the thread is considered as interactive. The niceness value of 0 is corresponding to spending more than 60% of the time sleeping (not including waiting time for CPU). That function is calculetad as fllowing:

$$scaling\ factor = m = 50$$

$$penalty(r,s) = \begin{cases} \frac{m}{\frac{s}{r}} & s > r \\ \frac{m}{\frac{r}{s}} + m & otherwise \end{cases}$$

Otherwise the thread is classified as a batch. That means negative value of nicenes what means higher priority make it easier for a thread to be considered as interactive.

In both interactive and batch runqueues ther is a FIFO "first in first out" which is used when a thread is added to the runqueue, the scheduling algorithm insert the added thread at the end of the FIFO and its given an index by the thread priorety. Picking a thread to run will be done by taking the first thread in highest priority and non empty FIFO.

In my understanding FIFO is considered as a lists in list, all threads with priorety 0 will be added in first position (first list in the FIFO list) in the list and so on. For exemple if we have already 2 threads in it then we add one more thread with priorety 0 it will be added in the same position but will be executed when the 2 threads before are executed.

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

The CFS support the task preemption while in ULE is disabled which means that only threads of the kernel can preempt others.

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?

They found that the main defference between the ULE and CFS is in handling of batch threads. While the CFS want to be fair for all threads the ULE gives the priority to the interactive threads.

For database workloadss and NAS they defines the preformence of comparing the number of operations per seconds and for other applications they compared as “1/execution time”

They define it in this way because it’s that the scheudling algorithm has little influence on the most of workloads, that’s because most applications uses thrads that preform same work, which means that both CFS and ULE end up scheduling the threasds in a round-robin fashion.

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?

In both CFS and ULE the workload balancing periodiclly and placement of threads happens when threads are created or waken up. CFS depend on a complex load matrix that use a hierarchical load strategy to balance. and it runs every 4ms, while the ULE tries to be fair by evening out the number of threads on each core. Balancing in ULE happens less often than CFS and it ignores topology of the hardwear.

In Bouron et al. test shows that CFS is way faste in balancing load than the ULE, the ULE takes one thread to each core each blanacing workload and while it runs once each (0,5s – 1,5s) less often than the CFS, at the end it reach a perfect balancing state (if the threads are perdectly divisble by the number of cores) while the CFS never reaches a perfect balance because it only tries to balance the load between NUMA nodes when it has an enough big imbalaced (25% load different in practice).

Source reference:

All answers is reffered to the paper "The Battle of the Schedulers: FreeBSD ULE vs. Linux CFS"

Link: https://www.usenix.org/conference/atc18/presentation/bouron