Task3 Scheduler Report

Yizhou Shan, Wan-Eih Huang
<shan13, huan1031@purdue.edu>

Overall

We implemented SCHED_MYCFS class, finished Part A, Part B-1. We used stable 4.10.6 kernel, the diff-file is attached in top directory. We did not implement another RB-Tree, we reused kernel's implementation.

Our MYCFS class is modeled after original CFS except its scheduling group and migration code. The selection of next running task is based on timeline, which is measured by virtual runtime.

Noteworthy Features

Several noteworthy features in our implementation:

1. We patched the `kernel/sched/debug.c` to let `/proc/sched_debug` show information about MYCFS class at runtime.
2. A newly created task that inherit SCHED_MYCFS from its parent task, will have a bigger vruntime at beginning. Doing this help us to keep the time promise we made to the current task (parent), and will run the newly created task in a near future.
3. The sleeping latency that within a single `sysctl_sched_latency` will not be counted toward vruntime. Doing this help the sleeping process to get a chance to run very shortly.
4. Some cached buddies are used to help us pick next runnable task. For example, the `next` buddy is set in the `yield_to_task_mycfs()`, and the nearest `pick_next_entity()` will use this next buddy no matter what. This clearly improves the decision controlling process.
5. We added a new kernel command option: `mycfs_debug`. Passing this will enable the verbose printing of mycfs.
6. We added a new field `mycfs_limit` in `sched_entity` structure and a new system call `sched_setlimit()` for part B. However, we failed to submit ms1 so we have to submit this intermediate version first. We will finish part B later.

Conclusion

We learned a lot about scheduling in task 3. We examined the CFS code very carefully and ported almost the same logic except scheduling group and migration. The virtual runtime, RB-tree based timeline design is straightforward but amazing. Especially the wall-time to virtual runtime calculation, it clearly says how the nice value impact the conversion.

Known Bugs

Let us say we have tasks A and B both in CFS class in the beginning. At runtime, we changed A from CFS to MYCFS class. Before A exits, there must be some schedule() invoked. Assume the `pick_next_task()` will choose B from CFS class to run next, then it will do context switch. After context switch, we will be in `finish_task_switch()`. And in this function, we can know that current is B and prev is A.

This whole process works well, and we expect that the B is running after context switch and the CPU can continue normally. But in our case, the CPU will be stuck with **IRQ enabled**. Kernel reports there are some **rcu_sched stalls**, which cause this CPU fail to proceed.

We tried hard to debug this but found nothing. The code path looks okay. After all, our workaround is: we add a check in CFS's picking function, if the previous task belongs to MYCFS, it will just return NULL and let MYCFS and IDLE class to run.