Task 3. Scheduler Final Report.

Yilun Chen(chen2709@purdue.edu)
Naman Nandan (nnandan@purdue.edu)

Overall Design:
We implemented *SCHED_MYCFS* scheduling policy, and sched_class called *mycfs_sched_class*, which is below *fair_sched_class* and above *idle_sched_class*. Our scheduler perform same behavior with Linux CFS except features about task migration, group scheduling, and multiprocessors scheduling.

We implemented on kernel 3.10.62, and implemented all callback functions except SMP and group scheduling related functions. Due to the sched_class standard has changed in later kernel, our implementation is not compatible with later kernels. We have done Part A but not implemented CPU usage limit (Part B). Our next ready-to picked task is determined by virtual runtime (execution time weighted by priority), the task with smallest vruntime is in RB tree leftmost Node would be scheduled.

We do not build our own RB-tree, and we are not building from scratch, since we port and taking advantages of some some low level CFS function (e.g. convert among task_struct, rq, mycfs_rq, checking if a task can preempt another by checking they vruntime interval, etc.)

Noteworthy Features:

1. By default, No one would fall into MYCFS class, and only use syscall sched_setschdeduler() can set a task to MYCFS class. After that all forked processes/threads would inherit MYCFS class.
2. By default, pick_next_task() would pick the leftmost Node on RB tree (with smallest vruntime to be scheduled). However, the exception is that *yield_to_task_mycfs()* is called by a process. At next time MYCFS scheduler making decision, if the leftmost task and yielded task do not differ much from each other in vruntime. The previous yield task would be picked.
3. Our test program can show the fairness of MYCFS, it first inits a counter with 0, and sets its scheduler as MYCFS scheduler. Then it do fork(), the child process would inherit parent's scheduler. After that both of them register a SIGALRM handler that print their current counter, and use setitimer to get a SIGALRM from kernel every 5s. After that they do infinite loop to increment counter. On uniprocessor, we expect they same amount of CPU time so that counter values are similar.
4. We add an marco DEBUG_MYCFS at beginning of mycfs.c which gives you some insight on what internally happen in MYCFS, but by default please comment out that line, otherwise the printk messages would overwhelm you.

Known Bugs:
In kernel/sched/core.c function wake_up_new_task, there is a line:
set_task_cpu(p, select_task_rq(p, SD_BALANCE_FORK, 0));

which is wrapped by #ifdef CONFIG_SMP, would corrupt a new forked process’ PC, since we do not consider SMP, I just commented it out.