CS692_Lab[3]

1: SCHED_FS

To implement fair sharing among users, I first looked at the implemented on CFS and how the algorithm works. For this, I did code reading for some of the important components of the CFS scheduler – task_struct, sched_entity, task_group, cfs_rq, rq and the role each of these play in the Red Black tree data structure.

So, for this lab, the most relevant concept is that CFS keeps track virtual runtime (vruntime) inside sched_entity to make scheduling decisions. It trying to evenly distribute the CPU time among tasks by trying to make vruntime equal for all tasks. This virtual runtime is not the actual time spent on CPU, that would be sum_exec_runtime. Vruntime is accumulated runtime manipulated by the “load” of each task. i.e. a task with less load will accumulate more runtime than a task with higher load. Any new task will be set to minimum vruntime on the tree. vruntime is cumulative. A value “delta”, which indicates the time it ran on current burst, is added at each update in the function `update_curr()`. This delta is what that is manipulated by weight at each burst and the weight can be modified from burst to burst.

```
delta = current burst * (1024/load)

vruntime += delta
```

Here, 1024 is the default load value which is divided by the user set load value. So, a task with load of 1024 will get twice as much actual runtime than a task with load 512 as since CFS tries to equal vunrtimes of all tasks.

So, I used this to implement SCHED_FS. For a user with n tasks, delta will be multiplied by n so sum of runtime of n tasks will be equal to runtime of another use with 1 tasks for whom delta is multiplied by 1.

To achieve this, I had to loop through all threads of all process with `for_each_process_thread(tg,t)` macro. I only count tasks that belong to same user and are in run state (`t->state`) i.e. ignore tasks that are sleeping. This way, I was able to achieve fair share among tasks dynamically.

I implemented the policy by hardcoding the UID range, so this policy is applied on top of other policies for UID >= 1001. I made a sysctl flag to enable or disable this scheduling behavior.

2: Profiling

For profiling, I made another sysctl entry to start or stop profiling. When stopped, the last captured profiling is dumped in logs which can be accessed by user using `dmesg` command. Before capturing profiling info again, `dmesg −c` must be used to clear old statistics.

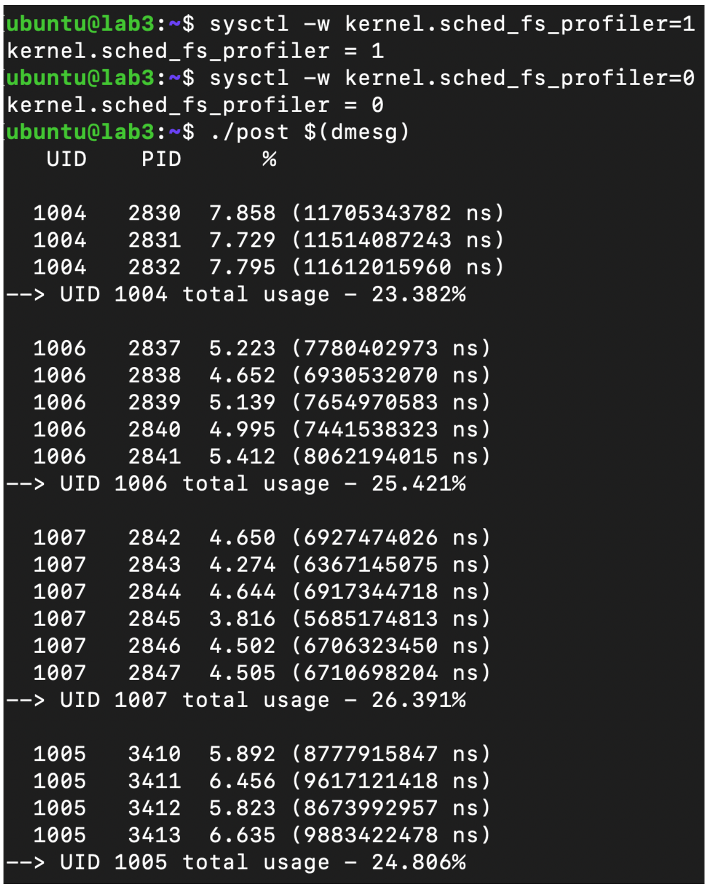
I am just capturing sum of exec time of each process which is done only when the profiling flag is enabled (by sysctl). And this is reset to zero when profiling is stopped.

3: Post processing

Once profiling is done, I wrote a program at user level to post process the data. This program simply prints usage for each process and then sum of those for each user. The program is run with the following command `./a.out $(dmesg)`

4: Result

I created for users A, B, C, and D and ran a simple while loop program. Then I enabled profiling for a minute. Then I post processed the data as shown below.



```
ubuntu@lab3:~$ sysctl -w kernel.sched_fs_profiler=1
kernel.sched_fs_profiler = 1
ubuntu@lab3:~$ sysctl -w kernel.sched_fs_profiler=0
kernel.sched_fs_profiler = 0
ubuntu@lab3:~$ ./post $(dmesg)
   UID    PID      %

  1004   2830  7.858 (11705343782 ns)
  1004   2831  7.729 (11514087243 ns)
  1004   2832  7.795 (11612015960 ns)
--> UID 1004 total usage - 23.382%

  1006   2837  5.223 (7780402973 ns)
  1006   2838  4.652 (6930532070 ns)
  1006   2839  5.139 (7654970583 ns)
  1006   2840  4.995 (7441538323 ns)
  1006   2841  5.412 (8062194015 ns)
--> UID 1006 total usage - 25.421%

  1007   2842  4.650 (6927474026 ns)
  1007   2843  4.274 (6367145075 ns)
  1007   2844  4.644 (6917344718 ns)
  1007   2845  3.816 (5685174813 ns)
  1007   2846  4.502 (6706323450 ns)
  1007   2847  4.505 (6710698204 ns)
--> UID 1007 total usage - 26.391%

  1005   3410  5.892 (8777915847 ns)
  1005   3411  6.456 (9617121418 ns)
  1005   3412  5.823 (8673992957 ns)
  1005   3413  6.635 (9883422478 ns)
--> UID 1005 total usage - 24.806%
```

Note:
If a user A has only one task and other user B has multiple, even though SCHED_FS is doing its job, user B will have an advantage on multicore system as his tasks are running in parallel.