

## EECS 388 Lab #8

# Playing with Linux Scheduler

In this lab, you will learn to interact with Linux's CPU schedulers and monitor their behaviors on Raspberry Pi 4. In the process, you will also learn to use several standard tools and scheduling related system call APIs.

## Part 1. Using scheduler related tools

On a terminal, create the following 'cpuhog' program and compile it.

```
$ cat cpuhog.c
int main()
{
    while(1);
}
```

```
$ gcc cpuhog.c -o cpuhog
```

Now, you shall use the 'taskset' utility to launch three instances of the 'cpuhog' program as follows.

```
$ taskset -c 0 ./cpuhog &
[1] 3361
$ taskset -c 0 ./cpuhog &
[2] 3378
$ taskset -c 0 ./cpuhog &
[3] 3379
```

The taskset utility controls which CPU core (cores) to execute the given program. In this case, it forces to schedule at core 0 (due to "-c 0").

On the terminal, execute the 'htop' program. You will see something like the following. (Alternatively, you can use 'top' program).

```

File Edit Tabs Help

 1 [|||||||||||||||||||||100.0%] Tasks: 84, 182 thr; 4 running
 2 [||| 3.9%] Load average: 3.22 1.77 0.77
 3 [|| 2.6%] Uptime: 04:44:48
 4 [|| 2.7%]
Mem[|||||||||||||||||695M/1.89G]
Swp[ 0K/100.0M]

  PID USER      PRI  NI  VIRT   RES   SHR  S  CPU% MEM%   TIME+  Command
 3361 pi          20   0   1720   304   248  R  33.7  0.0   0:39.89 ./cpuhog
 3379 pi          20   0   1720   320   264  R  33.7  0.0   0:38.36 ./cpuhog
 3378 pi          20   0   1720   324   268  R  33.0  0.0   0:38.64 ./cpuhog
 4044 pi          20   0   116M 30220 24224 S  2.0  1.5   0:00.61 gnome-screenshot
 6708 root        20   0   247M 56280 38392 S  2.0  2.8   3:13.18 /usr/lib/xorg/Xor
 1731 pi          20   0   8184  2860  2288  R  1.3  0.1   0:07.38 htop
 2867 pi          20   0   501M  138M 61124 S  0.7  7.2   0:46.93 /usr/lib/chromium
 1077 pi          20   0   558M  132M 84164 S  0.7  6.8   0:47.75 /usr/lib/chromium
 6727 root        20   0   247M 56280 38392 S  0.7  2.8   0:01.53 /usr/lib/xorg/Xor
 2161 pi          20   0   425M  144M  103M S  0.7  7.4   0:09.02 /usr/lib/chromium
 1142 pi          20   0   365M 80332 58148 S  0.0  4.0   0:40.30 /usr/lib/chromium
 1121 pi          20   0   558M  132M 84164 S  0.0  6.8   0:14.22 /usr/lib/chromium
 1218 pi          20   0   289M 73288 38932 S  0.0  3.7   0:08.43 /usr/lib/chromium
  824 pi          20   0 64476 16936 12968 S  0.0  0.9   0:00.74 openbox --config-
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice -F8Nice +F9Kill F10Quit

```

Next, you will change cpuhog instances' nice values (i.e., CFS priority values) using the 'renice' tool. Open up a new terminal so that we can keep monitoring the output of the top.

Check the PID values of the three cpuhog instances shown on the top screen. In the example above, they are 3379, 3378, 3361. Your PID values may be different. You can also check the pid values by using the 'pidof' tool as follows.

```

$ pidof cpuhog
3379 3378 3361

```

Now, let's change the nice value of the first cpuhog instance.

```

$ renice 5 3379

```

On the terminal executing htop, monitor the CPU utilization the cpuhog instances. You should see that the CPU utilization of the reniced cpuhog is dropped to around 14% while each of the other two cpuhog instances utilizes around 43%.

```

  PID USER      PR  NI  VIRT   RES   SHR  S  %CPU  %MEM   TIME+  COMMAND
 3378 pi          20   0   1720   324   268  R  43.0  0.0   1:54.44 cpuhog
 3361 pi          20   0   1720   304   248  R  42.7  0.0   1:55.71 cpuhog

```

```
3379 pi          25   5   1720   320   264 R  13.9   0.0   1:25.38 cpuhog
```

Next, you again change the second cpuhog instance's nice value as follows.

```
$ renice 5 3378
```

Then, again monitor the 'htop' screen. You should see each of the reniced cpuhog instances (3379 and 3378) consumes around 20% and the remaining 'normal' instance is consuming around 60%.

```
  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
3361 pi        20   0   1720   304   248 R  60.3   0.0   2:20.85 cpuhog
3379 pi        25   5   1720   320   264 R  19.9   0.0   1:33.62 cpuhog
3378 pi        25   5   1720   324   268 R  19.5   0.0   2:15.20 cpuhog
```

```
$ renice 5 3361
```

Monitor the 'htop' screen. You should now see all cpuhog instances equally share the CPU---33% each.

```
  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
3379 pi        25   5   1720   320   264 R  33.6   0.0   1:45.66 cpuhog
3361 pi        25   5   1720   304   248 R  33.2   0.0   2:47.86 cpuhog
3378 pi        25   5   1720   324   268 R  33.2   0.0   2:27.27 cpuhog
```

## Part 2. Using scheduler related system calls

So far, you have used 'taskset' and 'renice' tools to control cpu core and nice values of your cpuhog program. Now, instead of using these external tools, you need to modify the cpuhog.c code and directly use system calls.

First, modify cpuhog.c to always be scheduled on core 0. You need to use 'sched\_setaffinity' system call. See the manual.

```
$ man sched_setaffinity
```

Then, use the 'setpriority' system call to change the task's nice value to 5. Again, see the manual for usage.

```
$ man setpriority
```

**Save the modified code as lab8.c and show it to your TA.**