Studio7 Report

STUDIO7

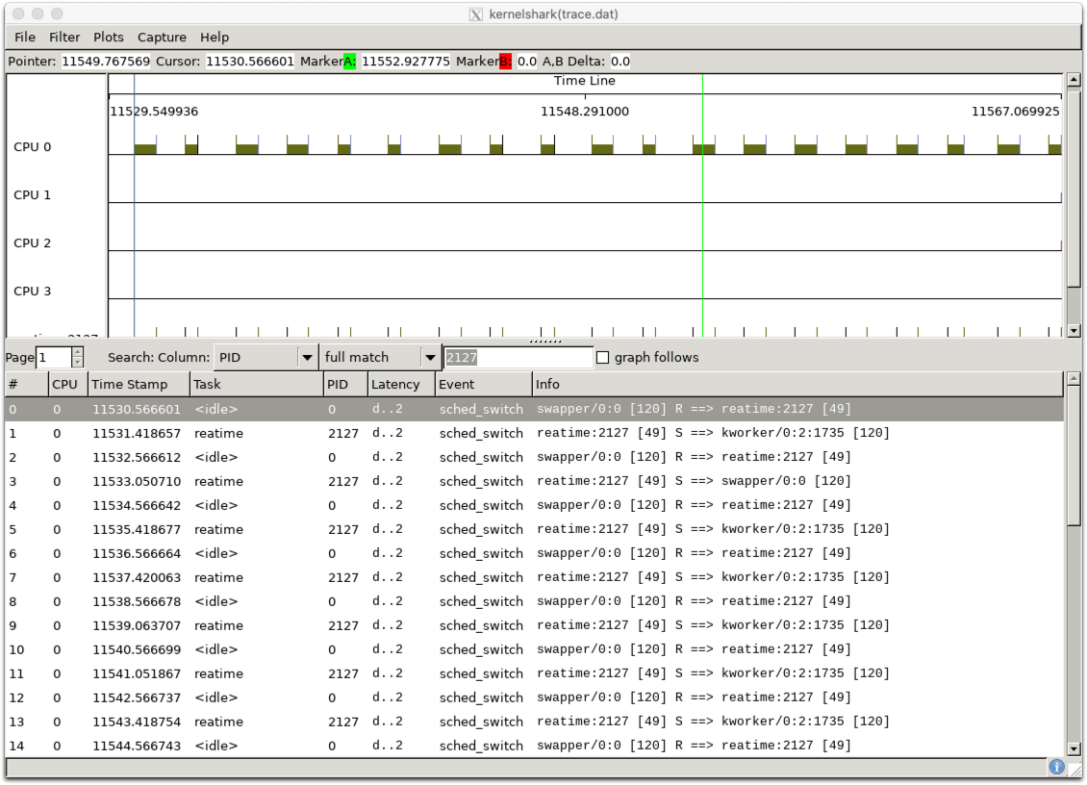
Step1:

- Bingxin Liu

Step2:

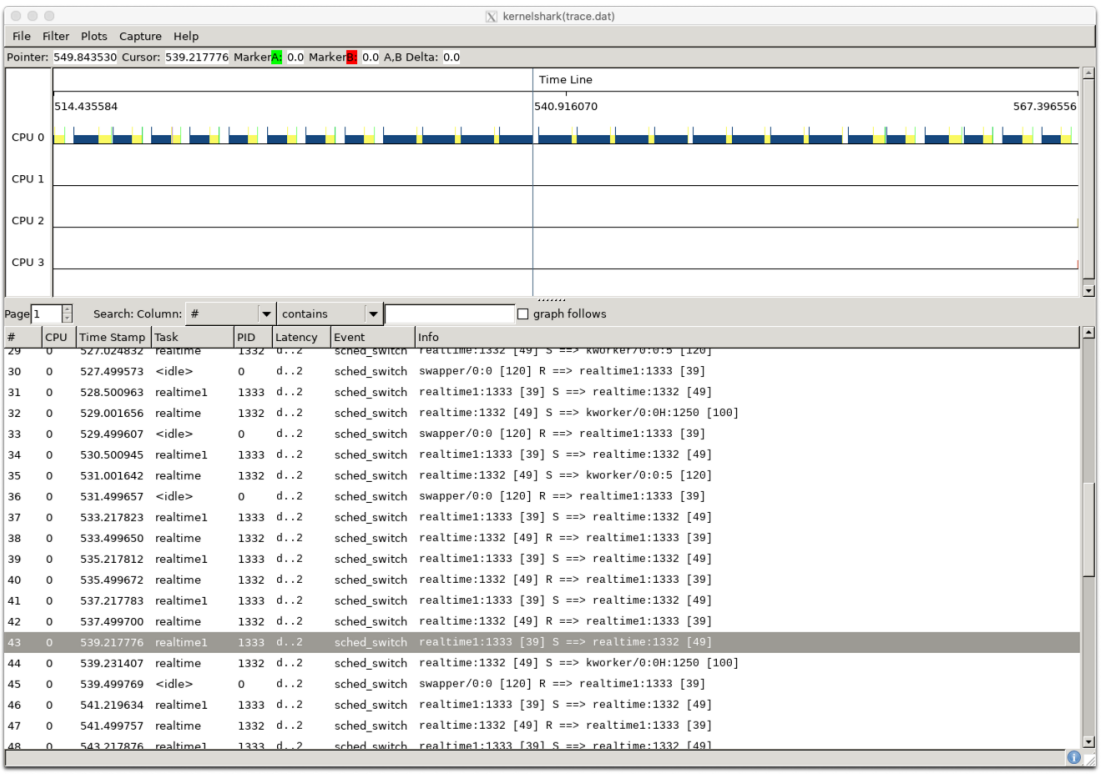
```
pi@lbxpi:~/modules $ sudo dmesg --clear
pi@lbxpi:~/modules $ sudo insmod realtime_module.ko period_sec=0 period_nsec=1000000000
pi@lbxpi:~/modules $ sudo rmmod realtime_module
pi@lbxpi:~/modules $ dmesg
[ 6759.185569] Loaded realtime module
[ 6759.185812] kernel thread reatime created.
[ 6760.185870] The kthread has just woken up
[ 6761.185883] The kthread has just woken up
[ 6762.185862] The kthread has just woken up
[ 6763.185876] The kthread has just woken up
//...
[ 6775.191663] Unloaded realtime module

pi@lbxpi:~/modules $ sudo dmesg --clear
pi@lbxpi:~/modules $ sudo insmod realtime_module.ko period_sec=2 period_nsec=0
pi@lbxpi:~/modules $ sudo rmmod realtime_module
pi@lbxpi:~/modules $ dmesg
[ 6834.016250] Loaded realtime module
[ 6834.016452] kernel thread reatime created.
[ 6836.016500] The kthread has just woken up
[ 6838.016523] The kthread has just woken up
[ 6840.016542] The kthread has just woken up
[ 6842.016595] The kthread has just woken up
[ 6844.016584] The kthread has just woken up
[ 6846.016630] The kthread has just woken up
//...
[ 6849.109855] Unloaded realtime module
pi@lbxpi:~/modules $

pi@lbxpi:~/modules $ sudo dmesg --clear
pi@lbxpi:~/modules $ sudo insmod realtime_module.ko period_sec=2 period_nsec=500000000
pi@lbxpi:~/modules $ sudo rmmod realtime_module
pi@lbxpi:~/modules $ dmesg
[ 7055.188207] Loaded realtime module
[ 7055.188421] kernel thread reatime created.
[ 7057.688485] The kthread has just woken up
[ 7060.188503] The kthread has just woken up
[ 7062.688526] The kthread has just woken up
[ 7065.188551] The kthread has just woken up
[ 7067.688572] The kthread has just woken up
[ 7070.188593] The kthread has just woken up
[ 7070.287487] The kthread has just woken up
[ 7070.287551] Unloaded realtime module
pi@lbxpi:~/modules $
```

Step3:



In my observation, I found that the kernel schedule our kernel thread every 2 seconds, which is the same as the time of period, which is 2 seconds (the number of iteration is 5,000,000). The accuracy is very good, the error is less than 1 ms at least. Basically, the ratio of workload of the simulated real-time task to period is one over four to one third. Sometimes, it takes longer time for the workload, which means longer than 0.5 second.

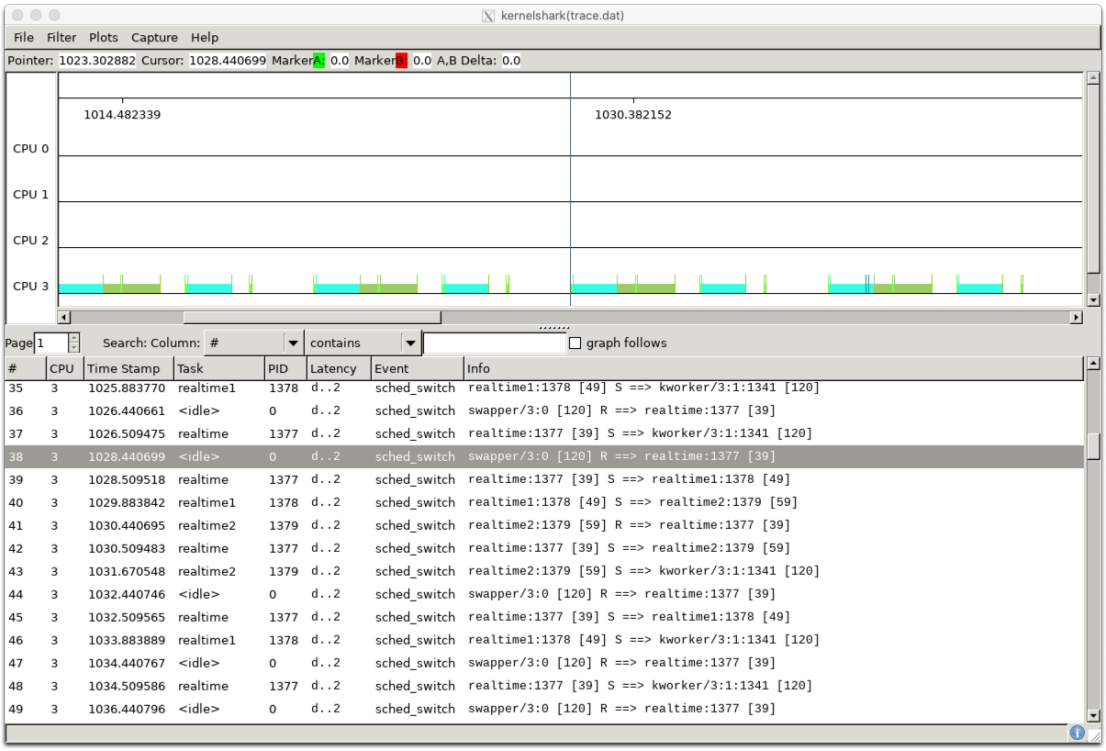
Step4:



In this time, at the begining, everthing goes well. However, at some point the kthread1 which has workload about 1 second and has higher priority than kthread which has worlaod about 0.5 second begins run slowly. As a result, there is not enough time for kthread to run before kthread preempted by kthread1 again. Ideally, the ratio of workload to period is two third, but actually it is about four fifth.

On the other side, the schedule works pretty well. The kthread which has higher priority always be scheduled first.

Step5:

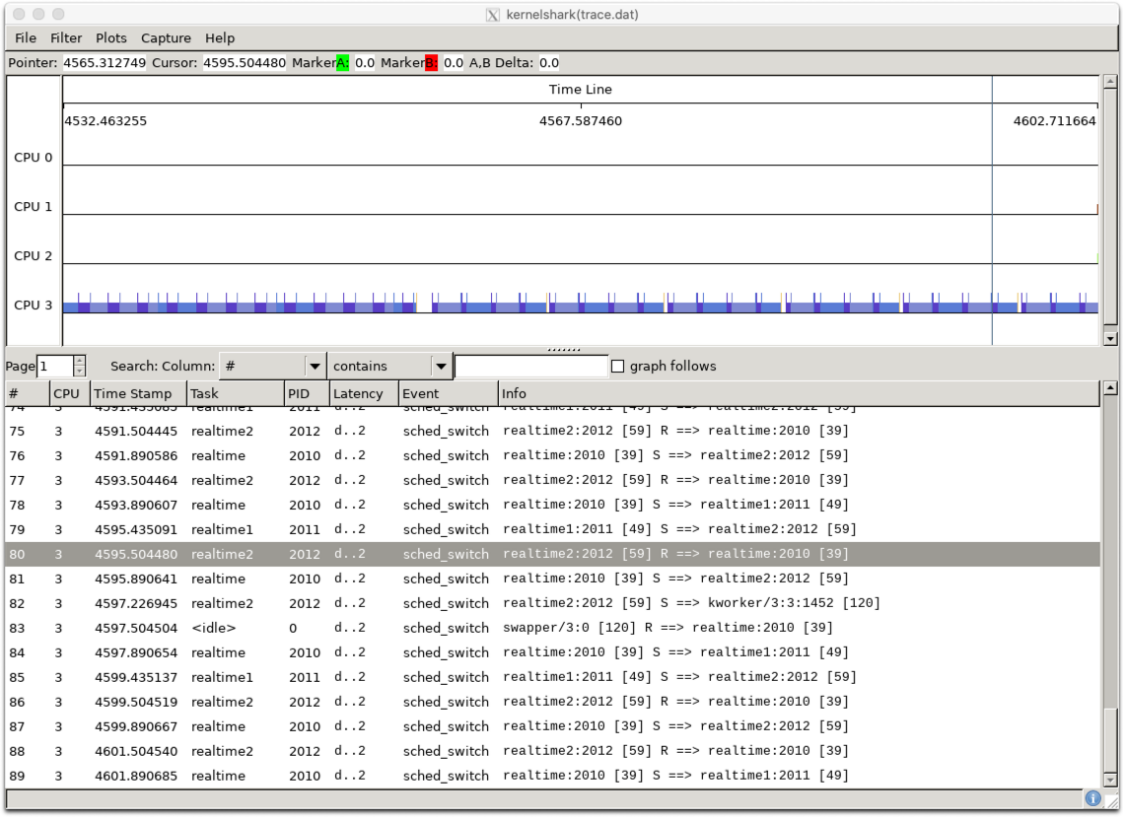


As we can see, basically, the kthread which has the highest priority will be scheduled first, and then the second highest, and finally the third. when the third kthread running, the first highest priority kthread woken up, and starts to run. At the same time, the third kthread stop, waiting the highest priority kthread finish, then the third finish the rest part. At the next time that the highest priority woken up, it runs then the second highest run. Because the lowest priority kthread has period four times more than the highest priority kthread, in this case, it will not be woken. Then the highest kthread is woken the third time. It runs and finish. After that, no ktrhead need run, so they are in the state of idle until the forth time the highest kthread woken up. The whole things then back to the start point and keep going.

Basically, the higher priority threads always scheduled before lower priority ones.

Also, the ratios of threads' workloads to its individual period are basically the same for each thread.

Step6:



In this time, we can find that between every hyperperiod, there is a little segment of "idle" time, which means we have basically gotten the best utilization.

20% individual period was utilized by the highest priority kthread.

40% utilization of indiviual period was used by the second highest priority kthread.

And also 40% untilization of individual period was used by the lowest priority kthread.

Sum up, we got nearly 100% untilization, which means almost all of hyperperiod is used.