1. As the answer to the first exercise, list the names of the people who worked together on this studio.

Zhe Wang, Jiangnan Liu, Qitao Xu

2. As the answer for this exercise, please paste fragments of the system log that show your kernel thread being woken up by the timer for different parameter values.

Our kernel thread is woken up every 1 second and every 2 seconds. As shown in the log below.

[ 516.778412] simple module initialized

[ 517.828438] In thread1

[ 518.878442] In thread1

[ 519.928436] In thread1

[ 520.978447] In thread1

[ 522.028432] In thread1

[ 523.078431] In thread1

[ 524.128445] In thread1

[ 525.178444] In thread1

[ 526.228431] In thread1

[ 527.278423] In thread1

[ 528.328441] In thread1

[ 528.806037] In thread1

[ 528.806172] Thread stopped

[ 528.806181] simple module is being unloaded

[ 535.760104] simple module initialized

[ 537.810132] In thread1

[ 539.860134] In thread1

[ 541.910166] In thread1

[ 543.960163] In thread1

[ 546.010151] In thread1

[ 548.060125] In thread1

[ 550.110121] In thread1

[ 552.160151] In thread1

[ 554.210141] In thread1

3. As the answer for this exercise, please briefly describe (based on what you see in the Kernelshark screenshot) what the scheduling behavior of the kernel thread looks like. What is the value of your chosen period, and how accurately is that period produced in the trace? Approximately what fraction of that period is occupied by the workload of the simulated real-time task (i.e., the amount of time spent in the inner loop relative to the period)?

schedule\_switch of "thread1" repeats in a period of about 1 second.

The chosen period is 1 second, and the margin of error is 0.001% ~ 0.0001%.

Approximately 20% (1000000 iterations) of that period is occupied by the workload of the simulated real-time task.

4. As the answer for this exercise, please summarize what you saw in the Kernelshark screenshot. Does the schedule look like what you expected? Is the thread with higher priority always scheduled before the lower priority one? What were the workloads of the two real-time tasks, compared to the chosen period?

schedule\_switch of "thread1" and "thread2" repeats in a period of about 1 second. Thread2 with a larger workload (1000000 iterations) uses more time to execute. For each period, thread2 with a higher priority is scheduled before thread1.

Yes, it almost looks like I exepected except for the slightly unstable running time of the workload in each period.

According to my experiment by exchanging the priority of thread1 and thread2, the thread with higher priority is always scheduled before the lower priority one.

Thread1 (1000000 iterations): Approximately 15% of that period is occupied by the workload of the simulated real-time task.

Thread2 (200000 iterations): Approximately 2%~4% of that period is occupied by the workload of the simulated real-time task.

5. As the answer for this exercise, please summarize what you saw in the Kernelshark screenshot. What is similar about the schedule compared to the previous one? Are higher priority threads always scheduled before lower priority ones? What were the workloads of the three real-time tasks, compared to their individual periods?

schedule\_switch of "thread3" (low priority: 15) repeats in a period of about 4 second. schedule\_switch of "thread2" (mid priority: 20) repeats in a period of about 2 second. schedule\_switch of "thread1" (high priority: 25) repeats in a period of about 1 second.

Yes, it is similar to the previous one. The thread with higher priority is always scheduled before the lower priority one.

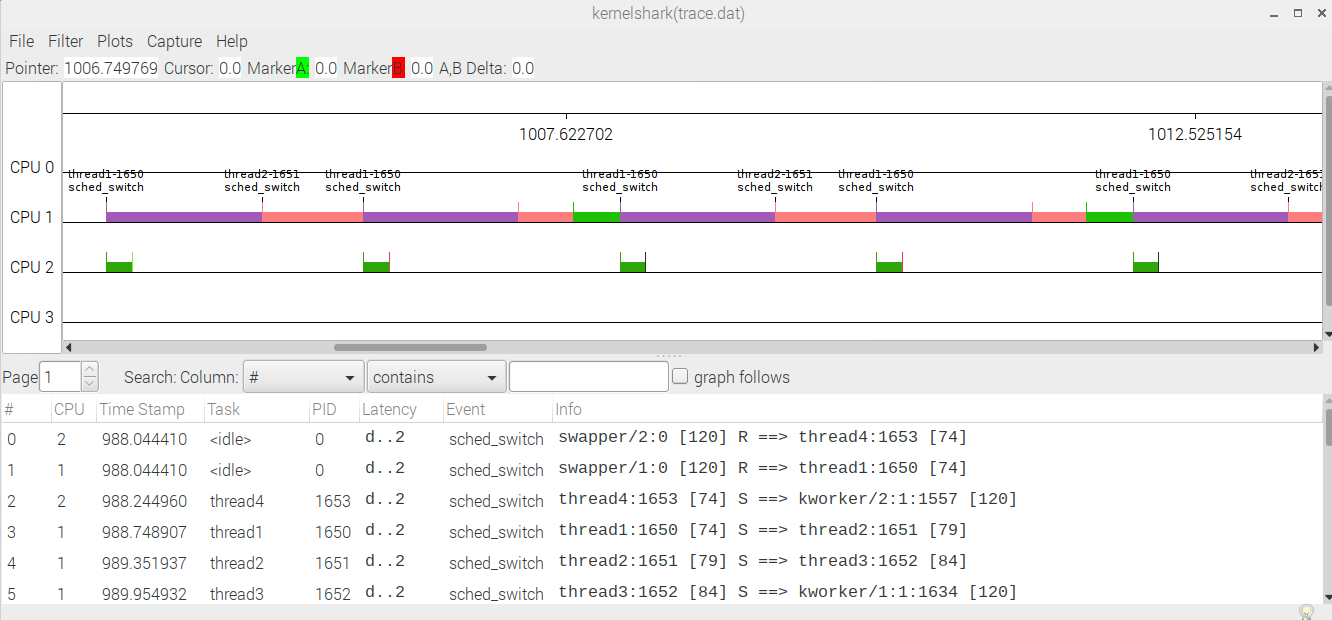
Thread1 (400000 iterations): Approximately 8% of that period is occupied by the workload of the simulated real-time task.

Thread2 (400000 iterations): Approximately 8% of that period is occupied by the workload of the simulated real-time task.

Thread3 (400000 iterations): Approximately 8% of that period is occupied by the workload of the simulated real-time task.

6. As the answer for this exercise, please summarize what you saw in the Kernelshark screenshot. What fraction of its individual period was each task able to utilize, and what fraction of the hyperperiod (in this case the period of the lowest priority task) was fully utilized by the set of tasks (versus being idle)?

Period is 2 seconds for each run. Hyperperiod is 8 seconds.

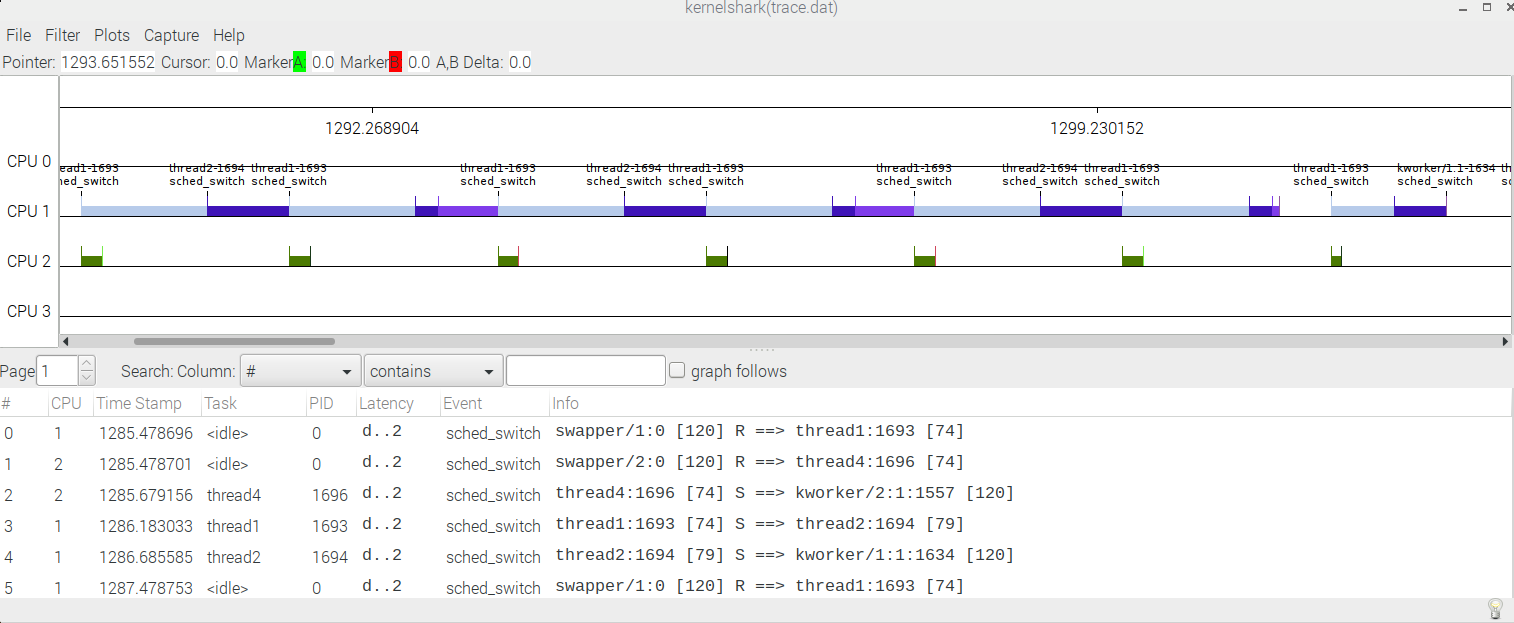


Thread 1. 6\*10^5

Thread 2. 6\*10^5

Thread 3. 6\*10^5

In this case, thread3 misses its first deadline. We need to adjust.

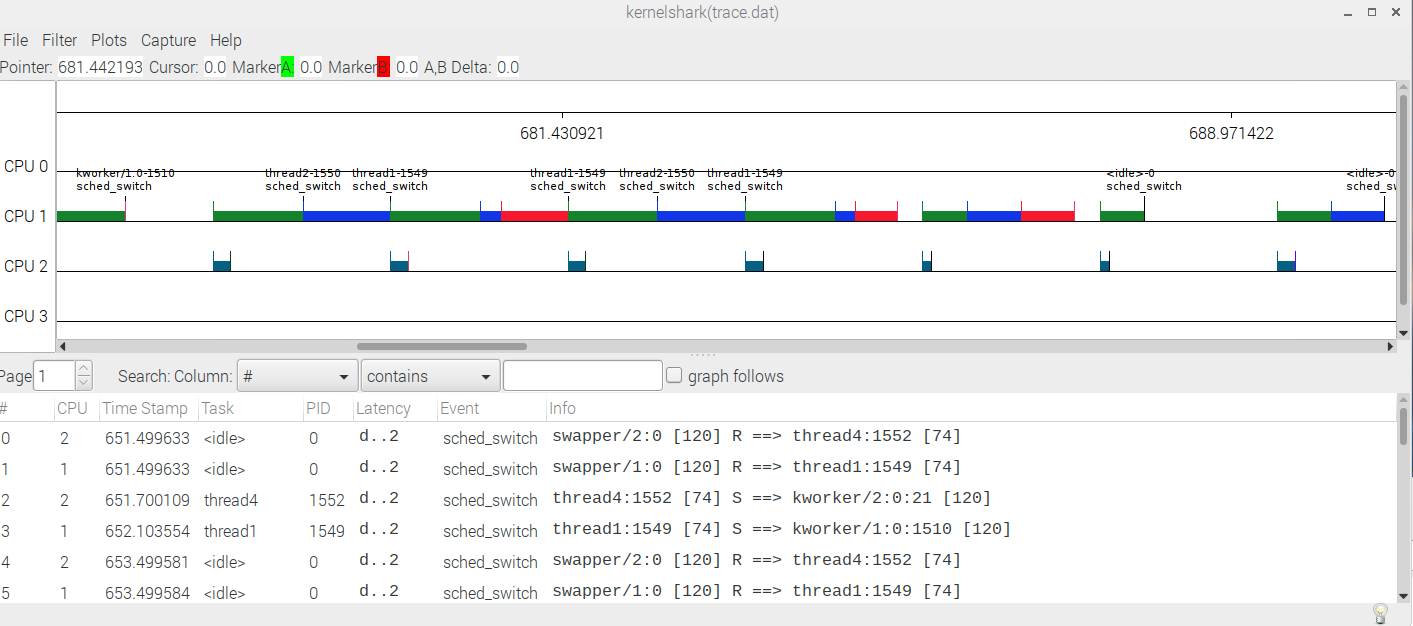


Thread 1. 6\*10^5

Thread 2. 5\*10^5

Thread 3. 6\*10^5

In this case, thread3 misses its first deadline. We need to adjust thread 1’s task time.

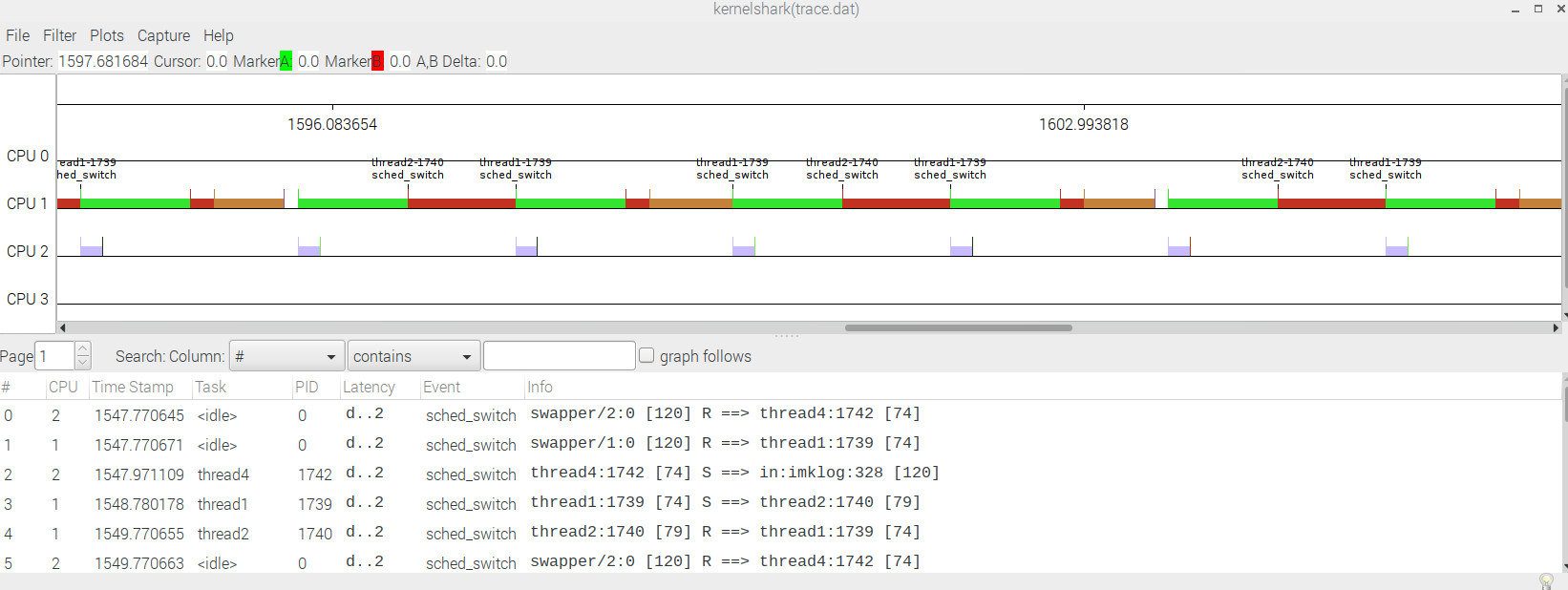


Thread 1. 5\*10^5

Thread 2. 6\*10^5

Thread 3. 6\*10^5

In this case, every thread finishes before its deadline. We can increase thread3’s work load a little bit.



Thread 1. 5\*10^5

Thread 2. 6\*10^5

Thread 3. 7\*10^5

The idle time in this scenario is pretty short. I think we have found the optimized workload distribution between the three threads.