Delft University of Technology

Real-time Systems
IN4343

Assignment II

Authors:

Jia Shi (5218845)
Kewei Du (5209633)

March 28, 2021

1 Question 1

1.1 General scheduling policy in Linux

Scheduling in Linux kernel indicates the concept that how the kernel schedules the threads to be run. Linux kernel implements the fixed-priority real-time scheduling, following the POSIX standard. Scheduling algorithms are preempted, otherwise the resources will always go starve since CPU is not released. A thread with lower priority will be preempted and returned to the wait queue in its priority level when a thread with higher priority arrives. The order of ready threads in equal priority based on the scheduling policy.

In this lab, we are required to in detail introduce three policies **SCHED_FIFO**, **SCHED_RR** and **SCHED_DEADLINE**, respectively.

1.2 SCHED_FIFO

SCHED_FIFO is a simple scheduling algorithm without time slicing.

Rule 1: When a thread with policy **SCHED_FIFO** is ready, it will be inserted into the end of queue for its priority.

Rule 2: Once a **SCHED_FIFO** thread in execution being preempted by another thread of higher priority, this thread will be swapped to the head of the queue for its priority. It will resume execution as soon as all threads of higher priority are done.

The priority of threads can be changed during its execution, or when it is ready. Once the priority is changed, thread with **SCHED_FIFO** in queue will alter its position accordingly. If raised, the thread will be placed at the end of the list for its new priority. If lowered, it is placed at the front of the list for its new priority.

1.3 SCHED_RR

SCHED_RR is same with the round robin algorithm we learned from the class. It's an expand for **SCHED_FIFO**: the ready queue is served with **SCHED_FIFO**, however, each task is executed under a maximum time quantum. If a **SCHED_RR** have been running for a time period same as the maximum time quantum, it will be sent back to the end of the ready list.

If a **SCHED_RR** thread has been preempted by a priority thread, it will complete the exceeded part of it later.

1.4 SCHED_DEADLINE

SCHED_DEADLINE is implemented using (Global Earliest Deadline First) in conjunction with CBS(Constant Bandwidth Server), it execute the task with the earliest absolute deadline.

Since the scheduling methods do not estimate any task other than the deadline of tasks, the situation that **SCHED_DEADLINE** thread is unfeasible has to be avoided. An admission test should be performed, if the change is unfeasible, scheduling methods fails with error **EBUSY**.

In order to guarantee the **SCHED_DEADLINE** policy, **SCHED_DEADLINE** threads have the highest priority in the system, it will preempt any other scheduled thread. The CBS guarantees non-interference between tasks, the threads that over-run their specified Runtime will be killed.

2 Question 2

2.1 Result for FIFO

Setting up: we used five threads, from thread 0 to 5, with priorities 25, 50, 50, 50, 90 respectively, and the periods 32000μs, 32000μs, 32000μs, 32000μs, 64000μs respectively.

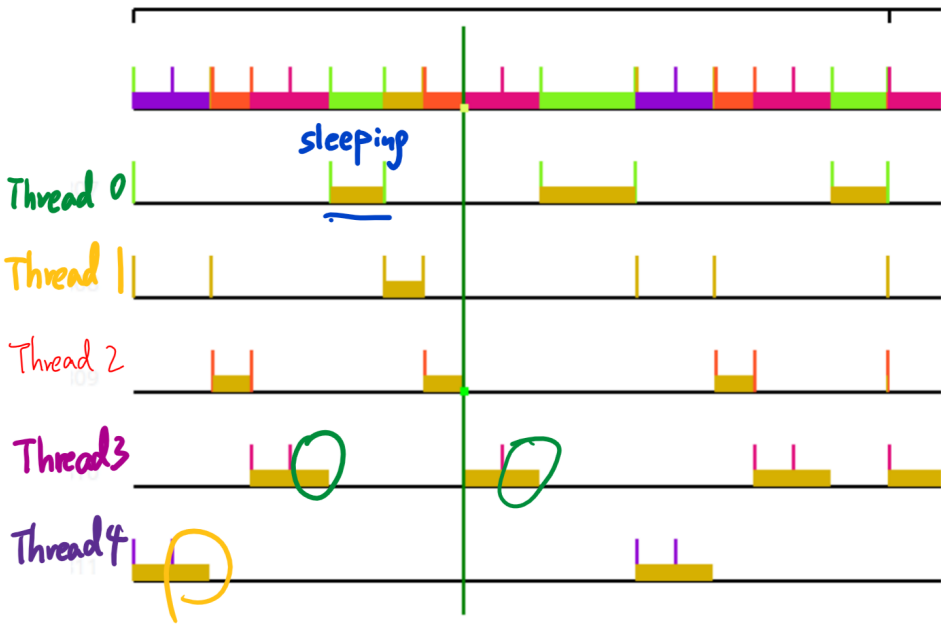

Figure 1: The output of FIFO scheduling

#	CPU	Time Stamp	Task	PID	Latency	Event	Info
9	1	20551.372257	<...>	14069	0	print	tracing_mark_write: RTS_Thread_4 Policy:FIFO Priority:99
10	1	20551.372258	<...>	14069	0	print	tracing_mark_write: RTS_Thread_4 Job 4 arrived at Time: 1616621347179283
11	1	20551.377259	<...>	14069	0	print	tracing_mark_write: RTS_Thread_4 Terminated ... ResponseTime:0 Deadline:0
12	1	20551.382190	<...>	14066	0	print	tracing_mark_write: RTS_Thread_1 Terminated ... ResponseTime:0 Deadline:0
13	1	20551.382200	<...>	14067	0	print	tracing_mark_write: RTS_Thread_2 Policy:FIFO Priority:50
14	1	20551.382202	<...>	14067	0	print	tracing_mark_write: RTS_Thread_2 Job 4 arrived at Time: 1616621347189227
15	1	20551.387207	<...>	14067	0	print	tracing_mark_write: RTS_Thread_2 Terminated ... ResponseTime:0 Deadline:0
16	1	20551.387216	<...>	14068	0	print	tracing_mark_write: RTS_Thread_3 Policy:FIFO Priority:50
17	1	20551.387218	<...>	14068	0	print	tracing_mark_write: RTS_Thread_3 Job 4 arrived at Time: 1616621347194243
18	1	20551.392230	<...>	14068	0	print	tracing_mark_write: RTS_Thread_3 Terminated ... ResponseTime:0 Deadline:0
19	1	20551.397225	<...>	14065	0	print	tracing_mark_write: RTS_Thread_0 Terminated ... ResponseTime:0 Deadline:0
20	1	20551.404089	<...>	14065	0	print	tracing_mark_write: RTS_Thread_0 Job 4 sleep finish: 1616621347211111

Figure 2: Tracing of FIFO scheduling

From the tracing window, we can see that the yellow circled part is running thread 1, the green part is running thread 0, while during the blue part, thread 0 is sleeping. As mentioned in 1.2, this scheduling method follows the first in first out scheduling method, thread 4 is always the first one to execute while thread0 is always the last one, because of their priority.

2.2 Result for RR

The setting of round robin scheduling method is same as the set up in 2.1. The thread with same priority(thread 1, thread 2, thread 3) will use round-robin method.

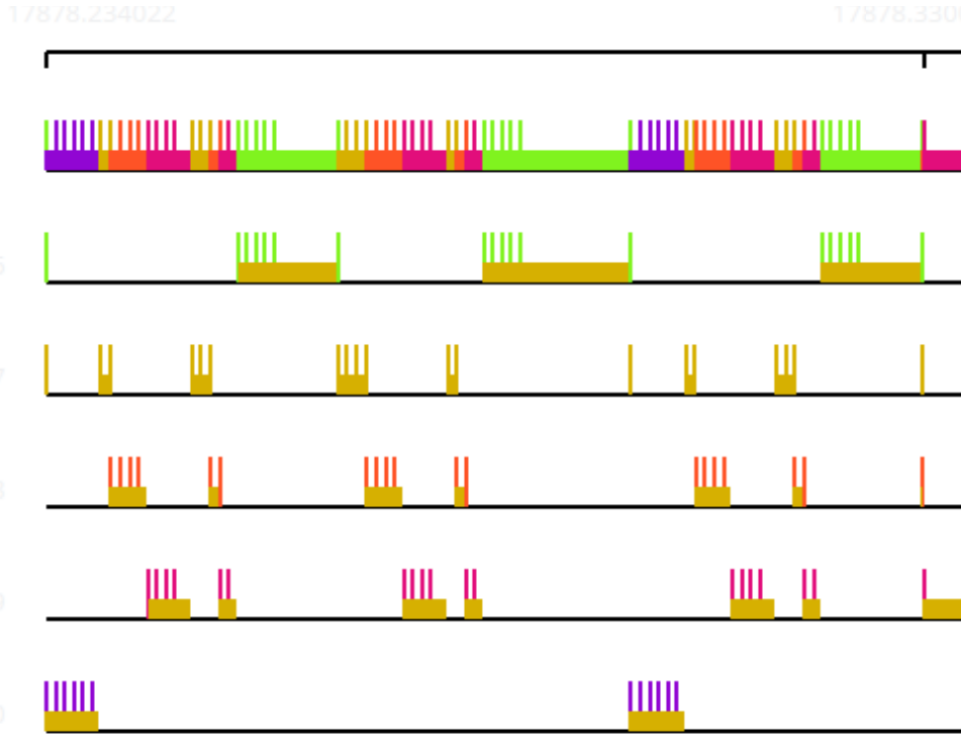

Figure 3: The output of RR scheduling

We set the ***timeslice*** as 4 milliseconds and the execution time for each thread is 5 milliseconds, we can see from picture 3, the threads can only execute for 4 milliseconds, the last part will be send back to the end of the list and execute later.

2.3 Result for OTHERS

The setting of other scheduling method is same as the set up in 2.1, only all the priority are set to 0, which is required by the scheduling option.

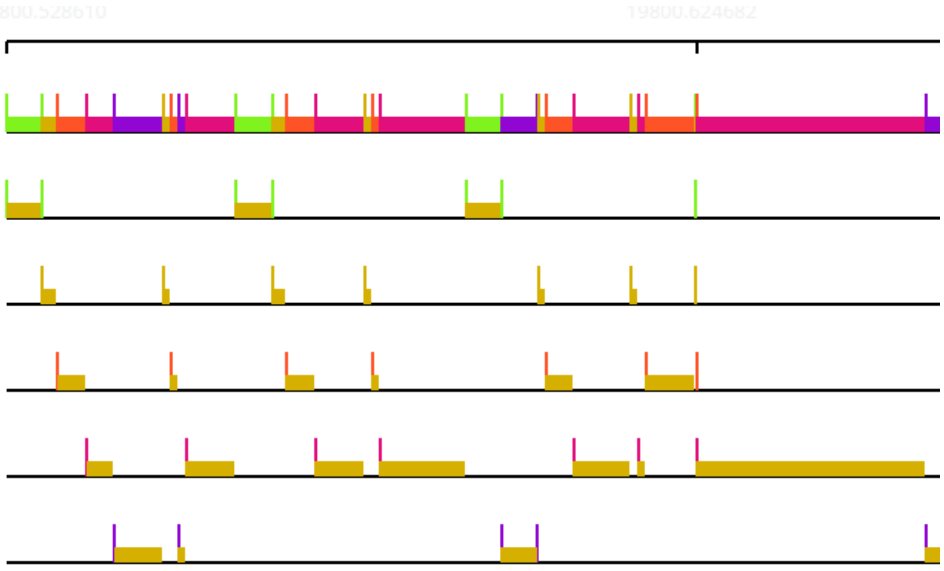

Figure 4: The output of OTHER scheduling

SCHED_OTHER randomly executes thread at the beginning, since all the threads have equal priority.

During the execution, the dynamic priority of running task will decrease while the priority of other tasks will increase. The dynamic priority is based on the nice value. This nice value is an attribute that can be used to influence the CPU scheduler to favor or disfavor a process in scheduling decisions. On Linux, the nice value is a per-thread attribute, which means different threads may have different nice values no matter whether they are in the same process or not.

This default Linux time-sharing scheduling which is activated by component **SCHED_OTHER** is designed for the threads which do not require the special real-time mechanisms and the number of nice values varies among UNIX systems, hence the result of **SCHED_OTHER** scheduling cannot be easily estimated.

3 Question 3

We already used the third method of building time-triggered tasks in Question 2, which is also the only accurate method to generate periodic tasks.

3.1 Result for method 1

From Figure 6, we can see that the tested period of thread 2 is $902903\mu s - 887884\mu s = 15019\mu s$, $14981\mu s$ smaller what we set in main function. This problem even makes the running sequence of thread different from the result of our setup. The reason for this problem is that the method does not consider any overhead like the terminating time and activating time.

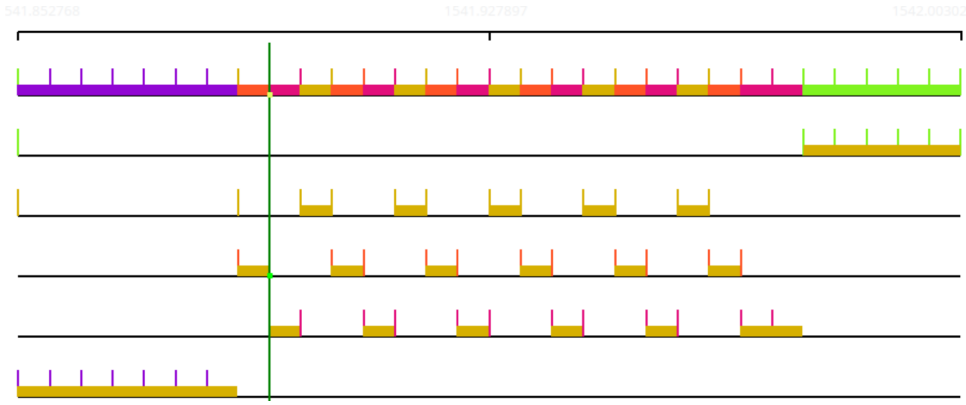

Figure 5: Result for method 1

Search: Column # contains Next Prev Graph follows

#	CPU	Time Stamp	Task	PID	Latency	Event	Info
21	1	1541.882958	<...>	25161	0	print	tracing_mark_write: RTS_Thread_4 Terminated ... ResponseTime:0 Deadline:0
22	1	1541.887874	<...>	25158	0	print	tracing_mark_write: RTS_Thread_1 Terminated ... ResponseTime:0 Deadline:0
23	1	1541.887883	<...>	25159	0	print	tracing_mark_write: RTS_Thread_2 Policy:FIFO Priority:50
24	1	1541.887884	<...>	25159	0	print	tracing_mark_write: RTS_Thread_2 Job 1 arrived at Time: 1616748625682562
25	1	1541.892886	<...>	25159	0	print	tracing_mark_write: RTS_Thread_2 Terminated ... ResponseTime:0 Deadline:0
26	1	1541.892893	<...>	25160	0	print	tracing_mark_write: RTS_Thread_3 Policy:FIFO Priority:50
27	1	1541.892893	<...>	25160	0	print	tracing_mark_write: RTS_Thread_3 Job 1 arrived at Time: 1616748625687572
28	1	1541.897895	<...>	25160	0	print	tracing_mark_write: RTS_Thread_3 Terminated ... ResponseTime:0 Deadline:0
29	1	1541.897900	<...>	25158	0	print	tracing_mark_write: RTS_Thread_1 Job 2 arrived at Time: 1616748625692576
30	1	1541.902901	<...>	25158	0	print	tracing_mark_write: RTS_Thread_1 Terminated ... ResponseTime:0 Deadline:0
31	1	1541.902903	<...>	25159	0	print	tracing_mark_write: RTS_Thread_2 Job 2 arrived at Time: 1616748625697581
32	1	1541.907904	<...>	25159	0	print	tracing_mark_write: RTS_Thread_2 Terminated ... ResponseTime:0 Deadline:0
33	1	1541.907907	<...>	25160	0	print	tracing_mark_write: RTS_Thread_3 Job 2 arrived at Time: 1616748625702585
34	1	1541.912910	<...>	25160	0	print	tracing_mark_write: RTS_Thread_3 Terminated ... ResponseTime:0 Deadline:0
35	1	1541.912913	<...>	25158	0	print	tracing_mark_write: RTS_Thread_1 Job 3 arrived at Time: 1616748625707591
36	1	1541.917915	<...>	25158	0	print	tracing_mark_write: RTS_Thread_1 Terminated ... ResponseTime:0 Deadline:0
37	1	1541.917918	<...>	25159	0	print	tracing_mark_write: RTS_Thread_2 Job 3 arrived at Time: 1616748625712597
38	1	1541.922920	<...>	25159	0	print	tracing_mark_write: RTS_Thread_2 Terminated ... ResponseTime:0 Deadline:0
39	1	1541.922923	<...>	25160	0	print	tracing_mark_write: RTS_Thread_3 Job 3 arrived at Time: 1616748625717602
40	1	1541.927925	<...>	25160	0	print	tracing_mark_write: RTS_Thread_3 Terminated ... ResponseTime:0 Deadline:0
41	1	1541.927928	<...>	25158	0	print	tracing_mark_write: RTS_Thread_1 Job 4 arrived at Time: 1616748625722607
42	1	1541.932931	<...>	25158	0	print	tracing_mark_write: RTS_Thread_1 Terminated ... ResponseTime:0 Deadline:0
43	1	1541.932934	<...>	25159	0	print	tracing_mark_write: RTS_Thread_2 Job 4 arrived at Time: 1616748625727612
44	1	1541.937936	<...>	25159	0	print	tracing_mark_write: RTS_Thread_2 Terminated ... ResponseTime:0 Deadline:0
45	1	1541.937939	<...>	25160	0	print	tracing_mark_write: RTS_Thread_3 Job 4 arrived at Time: 1616748625732618
46	1	1541.942942	<...>	25160	0	print	tracing_mark_write: RTS_Thread_3 Terminated ... ResponseTime:0 Deadline:0

Figure 6: Tracing of the first method

3.2 Result for method 2

The only difference between method 2 and method 3(right result) is that the sleep time of method 2 is an absolute time. The function of method 2 will be right if there is no preemption.

However, when the thread is preempted between calls to ***clock_gettime()*** and ***nanosleep()***, the period of thread will be wrong. As shown in Figure 8, the priority of threads is 4, 2, 2, 1 respectively, and 30000μs, 29500μs, 30000μs, 25000μs, 24000μs respectively. We can see a gap for thread 1 from task 2's execution to task 3's execution which is $26900\mu s - 791803\mu s = 34997\mu s$, $5997\mu s$ lager than expected.

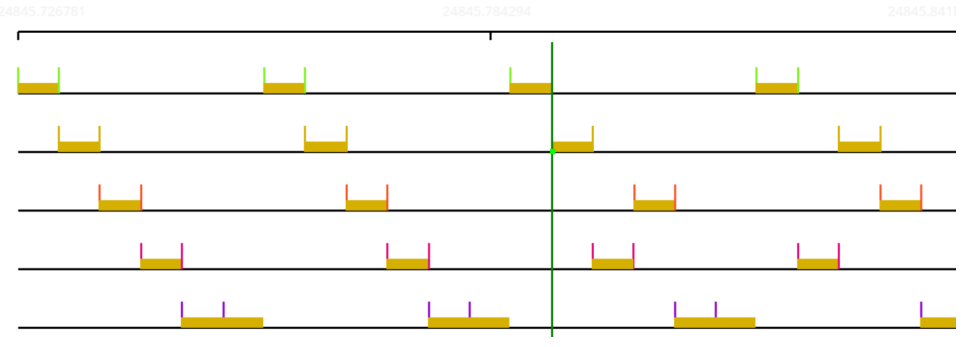

Figure 7: Result for method 2

#	CPU	Time Stamp	Task	PID	Latency	Event	Info
39	1	24845.781820	<...>	9745	0	print	tracing_mark_write: RTS_Thread_4 Job 1 Terminated ... ResponseTime:5 Deadline:0
40	1	24845.786797	<...>	9741	0	print	tracing_mark_write: RTS_Thread_0 Job 2 arrived at Time: 1616860380485123
41	1	24845.786799	<...>	9741	0	print	tracing_mark_write: RTS_Thread_0 start work ...
42	1	24845.791800	<...>	9741	0	print	tracing_mark_write: RTS_Thread_0 Job 2 Terminated ... ResponseTime:5 Deadline:0
43	1	24845.791803	<...>	9742	0	print	tracing_mark_write: RTS_Thread_1 Job 2 arrived at Time: 1616860380490131
44	1	24845.791804	<...>	9742	0	print	tracing_mark_write: RTS_Thread_1 start work ...
45	1	24845.796805	<...>	9742	0	print	tracing_mark_write: RTS_Thread_1 Job 2 Terminated ... ResponseTime:5 Deadline:0
46	1	24845.796810	<...>	9744	0	print	tracing_mark_write: RTS_Thread_3 Job 2 arrived at Time: 1616860380495138
47	1	24845.796811	<...>	9744	0	print	tracing_mark_write: RTS_Thread_3 start work ...
48	1	24845.801812	<...>	9744	0	print	tracing_mark_write: RTS_Thread_3 Job 2 Terminated ... ResponseTime:5 Deadline:0
49	1	24845.801815	<...>	9743	0	print	tracing_mark_write: RTS_Thread_2 Job 2 arrived at Time: 1616860380500143
50	1	24845.801816	<...>	9743	0	print	tracing_mark_write: RTS_Thread_2 start work ...
51	1	24845.806817	<...>	9743	0	print	tracing_mark_write: RTS_Thread_2 Job 2 Terminated ... ResponseTime:5 Deadline:0
52	1	24845.806823	<...>	9745	0	print	tracing_mark_write: RTS_Thread_4 Job 2 arrived at Time: 1616860380505151
53	1	24845.806824	<...>	9745	0	print	tracing_mark_write: RTS_Thread_4 start work ...
54	1	24845.811825	<...>	9745	0	print	tracing_mark_write: RTS_Thread_4 Job 2 Terminated ... ResponseTime:5 Deadline:0
55	1	24845.816794	<...>	9741	0	print	tracing_mark_write: RTS_Thread_0 Job 3 arrived at Time: 1616860380515121
56	1	24845.816795	<...>	9741	0	print	tracing_mark_write: RTS_Thread_0 start work ...
57	1	24845.821795	<...>	9741	0	print	tracing_mark_write: RTS_Thread_0 Job 3 Terminated ... ResponseTime:5 Deadline:0
58	1	24845.821798	<...>	9744	0	print	tracing_mark_write: RTS_Thread_3 Job 3 arrived at Time: 1616860380520126
59	1	24845.821798	<...>	9744	0	print	tracing_mark_write: RTS_Thread_3 start work ...
60	1	24845.826798	<...>	9744	0	print	tracing_mark_write: RTS_Thread_3 Job 3 Terminated ... ResponseTime:5 Deadline:0
61	1	24845.826800	<...>	9742	0	print	tracing_mark_write: RTS_Thread_1 Job 3 arrived at Time: 1616860380525128
62	1	24845.826800	<...>	9742	0	print	tracing_mark_write: RTS_Thread_1 start work ...
63	1	24845.831801	<...>	9742	0	print	tracing_mark_write: RTS_Thread_1 Job 3 Terminated ... ResponseTime:5 Deadline:0
64	1	24845.831802	<...>	9743	0	print	tracing_mark_write: RTS_Thread_2 Job 3 arrived at Time: 1616860380530131

Figure 8: Tracing of the second method

3.3 Result for method 3

Figure 2 shows scheduling result for the third method. From Figure 9 we can see that the tested period of thread 0 is $3571084\mu s - 539129\mu s = 31955\mu s$, almost same as what we set in main function.

#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	0	5609.539117	<...>	26129	0	print	tracing_mark_write: RTS_Spawned Thread_0
1	1	5609.539126	<...>	26130	0	print	tracing_mark_write: RTS_Thread_0 Policy:FIFO Priority:25
2	1	5609.539129	<...>	26130	0	print	tracing_mark_write: RTS_Thread_0 Job 4 arrived at Time: 1616752693379641
3	0	5609.539139	<...>	26129	0	print	tracing_mark_write: RTS_Spawned Thread_1
4	1	5609.539143	<...>	26131	0	print	tracing_mark_write: RTS_Thread_1 Policy:FIFO Priority:50
5	1	5609.539144	<...>	26131	0	print	tracing_mark_write: RTS_Thread_1 Job 4 arrived at Time: 1616752693379656
6	0	5609.539161	<...>	26129	0	print	tracing_mark_write: RTS_Spawned Thread_2
7	0	5609.539180	<...>	26129	0	print	tracing_mark_write: RTS_Spawned Thread_3
8	0	5609.539197	<...>	26129	0	print	tracing_mark_write: RTS_Spawned Thread_4
9	1	5609.539200	<...>	26134	0	print	tracing_mark_write: RTS_Thread_4 Policy:FIFO Priority:99
10	1	5609.539201	<...>	26134	0	print	tracing_mark_write: RTS_Thread_4 Job 4 arrived at Time: 1616752693379713
11	1	5609.544203	<...>	26134	0	print	tracing_mark_write: RTS_Thread_4 Terminated ... ResponseTime:0 Deadline:0
12	1	5609.549153	<...>	26131	0	print	tracing_mark_write: RTS_Thread_1 Terminated ... ResponseTime:0 Deadline:0
13	1	5609.549157	<...>	26132	0	print	tracing_mark_write: RTS_Thread_2 Policy:FIFO Priority:50
14	1	5609.549158	<...>	26132	0	print	tracing_mark_write: RTS_Thread_2 Job 4 arrived at Time: 1616752693389670
15	1	5609.554159	<...>	26132	0	print	tracing_mark_write: RTS_Thread_2 Terminated ... ResponseTime:0 Deadline:0
16	1	5609.554163	<...>	26133	0	print	tracing_mark_write: RTS_Thread_3 Policy:FIFO Priority:50
17	1	5609.554164	<...>	26133	0	print	tracing_mark_write: RTS_Thread_3 Job 4 arrived at Time: 1616752693394677
18	1	5609.559166	<...>	26133	0	print	tracing_mark_write: RTS_Thread_3 Terminated ... ResponseTime:0 Deadline:0
19	1	5609.564161	<...>	26130	0	print	tracing_mark_write: RTS_Thread_0 Terminated ... ResponseTime:0 Deadline:0
20	1	5609.571083	<...>	26130	0	print	tracing_mark_write: RTS_Thread_0 Job 4 sleep finish: 1616752693411593
21	1	5609.571084	<...>	26130	0	print	tracing_mark_write: RTS_Thread_0 Job 5 arrived at Time: 1616752693411597
22	1	5609.571123	<...>	26131	0	print	tracing_mark_write: RTS_Thread_1 Job 4 sleep finish: 1616752693411635
23	1	5609.571124	<...>	26131	0	print	tracing_mark_write: RTS_Thread_1 Job 5 arrived at Time: 1616752693411636
24	1	5609.576125	<...>	26131	0	print	tracing_mark_write: RTS_Thread_1 Terminated ... ResponseTime:0 Deadline:0
25	1	5609.576129	<...>	26132	0	print	tracing_mark_write: RTS_Thread_2 Job 4 sleep finish: 1616752693416642

Figure 9: Tracing of the third method