Real-time Tasks Models in Linux

Assignment 1 Report

CSE- 438
Embedded Systems Programming

by
Ritesh Thakur(1215283069)
Alisha Kulkarni(1215311903)

Under the guidance of
Prof. Yann-Hang Lee

The objective of this assignment is to be able to implement real-time tasks on Linux environment. The code is designed to run on the Ubuntu host platform as well as Galileo Gen2 board. It utilizes concepts of mutexes and priority scheduling so that all tasks are executed in a specific order and none of them are terminated abruptly.

In our code, we read specific inputs and implement the following functions:

- **periodic_tasks():** This function runs a task x number of times based on priority and in a specific time interval. This function is called by threads line2(), line3(), line4().
  Line2_thread() : This thread has 20 priority and waits on the conditional broadcast from Line3_thread(). Once it receives the broadcast it, does a conditional broadcast for line4()_thread.
  Line3_thread(): This thread had 10 priority and waits on the conditional broadcast from main(). Once it receives the broadcast it, does a conditional broadcast for line2() thread.
  Line4()_thread: This thread has 30 priority and wait on the conditional broadcast from line2().

- This technique is called “PREDICATING” and this ensures that threads are synchronised from the moment of their calling. One Condition broadcast sometimes results into deadlocks. It may happen that other threads get unblocked from their waiting condition and before their wait(), so they miss this signal forever. In some cases a low priority thread might acquire mutex before highest priority thread (because of other processes running on CPU, system clock granularity) and in that case least priority could execute before highest priority too.

- **aperiodic_tasks()**: This function runs the mouse task x number of times once mouse right/left click is released. The mouse_detection() thread calls this function. When you left release, aperiodic event 0 instance is called. Similarly when you right release, aperiodic event 1 instance is called. Priority for mouse_detect() thread is kept to 199 to ensure that it runs in the background all the time and doesn't make thread line4() wait on it.

- Once the **pthread_cond_broadcast()** is called from the main(), highest priority thread is activated and rest execute based on their priority. As shown in Illustration 1, the threads follow FIFO policy. The thread with the highest priority implements most often.

- The main() runs on any core till it encounters **pthread_set_affinity_np()** which ensures that all the threads run on CORE 0 thereafter. After all threads are killed, main finishes it execution on the core that it started on initially.

- Mutexes ensure that no thread interferes with the task completion of the previous thread. The code implements 4 threads which have a specific priority for implementation. All these threads need to be completed within a given time interval. To ensure no thread exits without completing the task we have implemented **pthread_setcancelstate()** functionality of Linux. This function for a thread is enabled only after execution of task body and just before it sleeps.

- Once the time interval ends, all threads are destroyed only and only once they have completed the task. This may lead to a few millisecond delay, to allow the completion of any pending task. We are using pthread_SETCANCELSTATE() to control this behaviour.

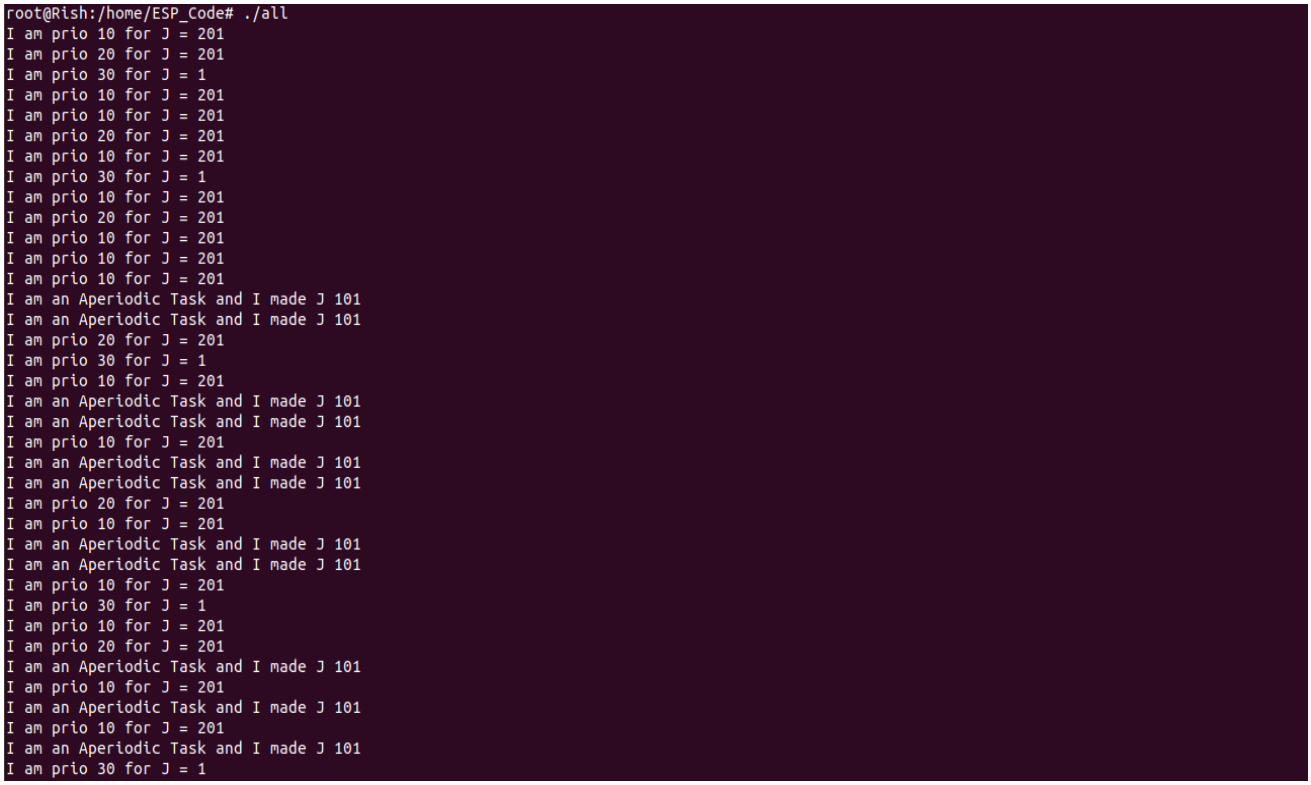


Illustration 1: Screenshot 1 of Terminal was taken before Logging was enabled to see output.

Illustration 2 shows us the kernelshark implementation. As shown, all threads are implementing on a single core.
Few observations from kernelshark:

all-26283: is main(). Main in this case runs a little bit over program's defined execution interval because pthread_setcancel() is waiting on one of the threads to finish its execution.

all-26284: aperiodic event which is triggered when you release the right or left click of mouse.

all- 26285: task with priority 10, highest priority, executes most often

all-26286: task with priority 20, which gets killed when it's asleep once the pthread_setcancelstate() is set to ENABLED

all-26287: task with priority 30, lowest priority and it gets killed while sleeping too.

All the events apart from the main() function execute on CPU0. CPU0 selects the threads based on priority.

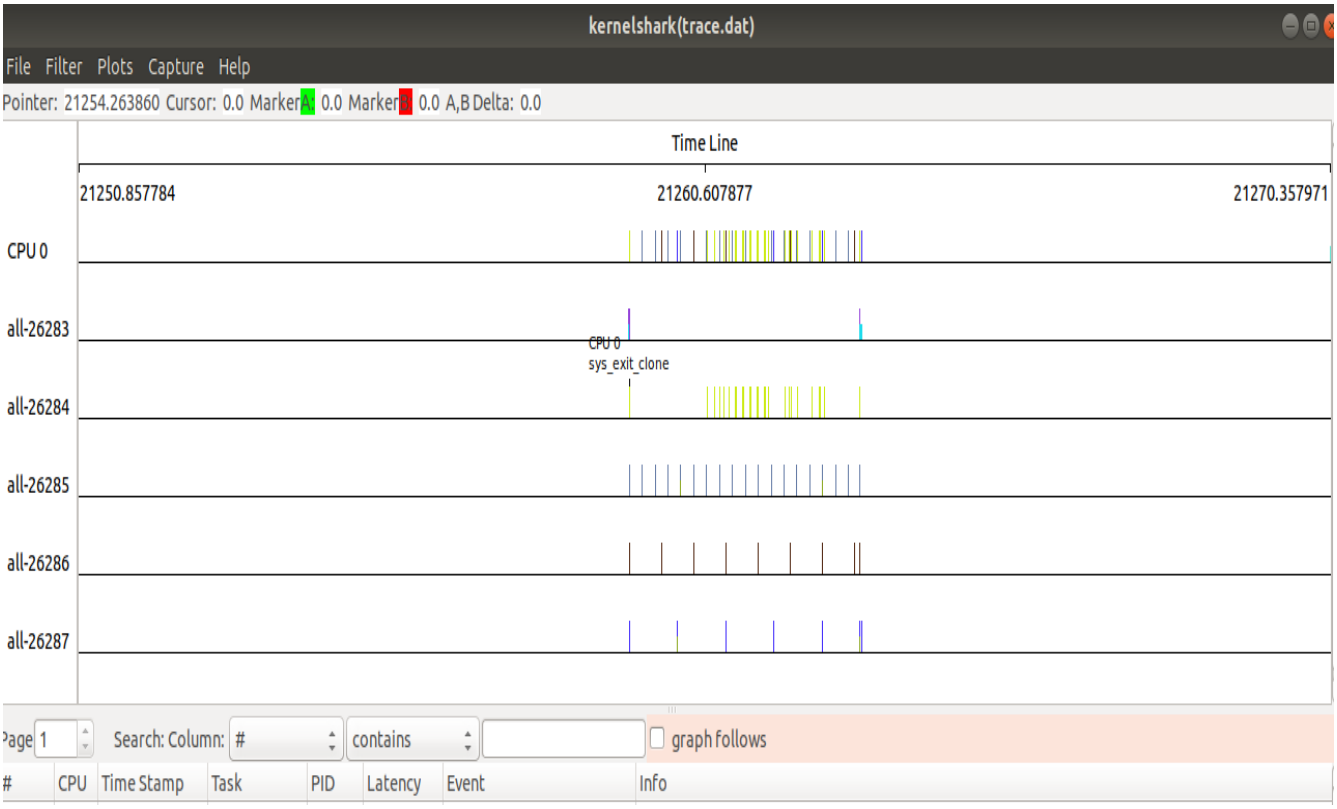
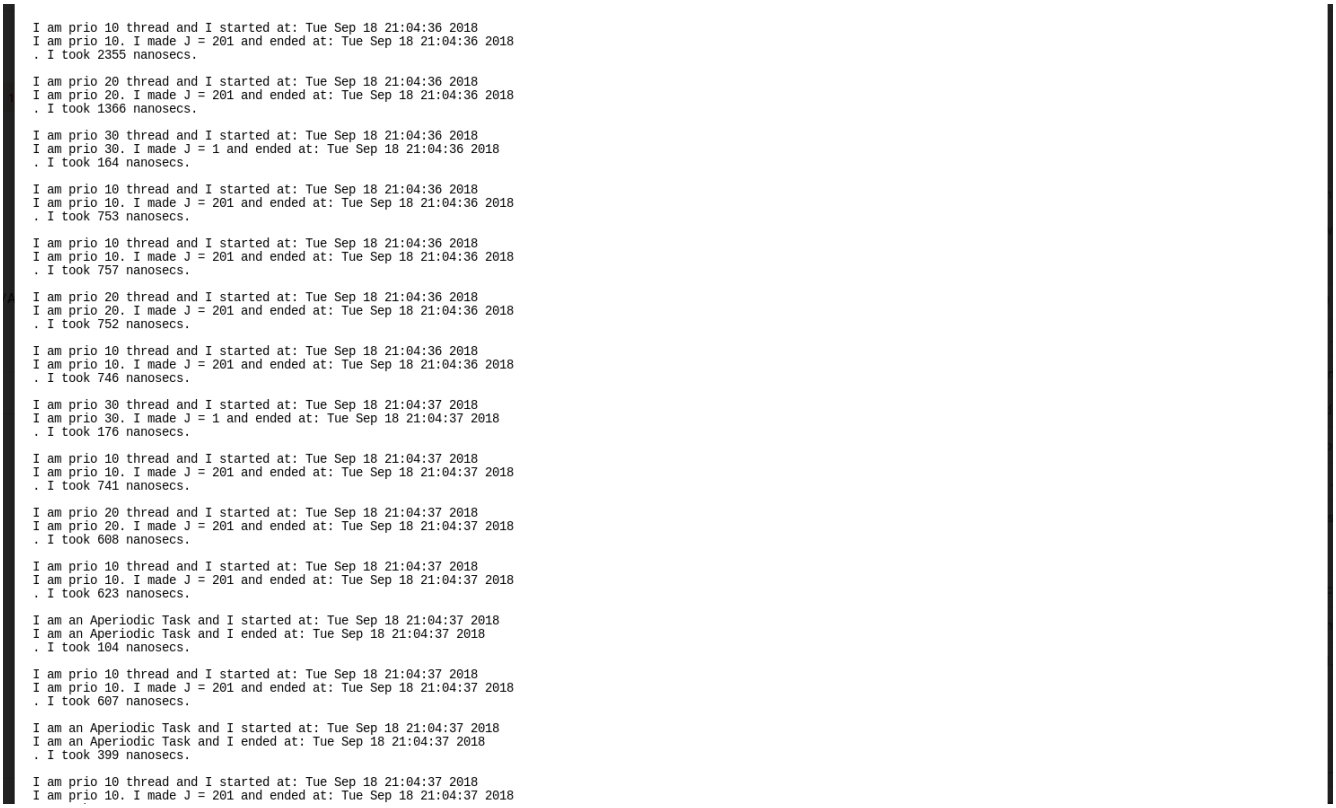


Illustration 2: Kernelshark

Illustration 3 shows an instance of Log.txt that stores the execution time of every thread. It stores the start and stop time of a task and it's elapsed time in nanoseconds. This was implemented using **clock_gettime()**



```
I am prio 10 thread and I started at: Tue Sep 18 21:04:36 2018
I am prio 10. I made J = 201 and ended at: Tue Sep 18 21:04:36 2018
. I took 2355 nanosecs.

I am prio 20 thread and I started at: Tue Sep 18 21:04:36 2018
I am prio 20. I made J = 201 and ended at: Tue Sep 18 21:04:36 2018
. I took 1366 nanosecs.

I am prio 30 thread and I started at: Tue Sep 18 21:04:36 2018
I am prio 30. I made J = 1 and ended at: Tue Sep 18 21:04:36 2018
. I took 164 nanosecs.

I am prio 10 thread and I started at: Tue Sep 18 21:04:36 2018
I am prio 10. I made J = 201 and ended at: Tue Sep 18 21:04:36 2018
. I took 753 nanosecs.

I am prio 10 thread and I started at: Tue Sep 18 21:04:36 2018
I am prio 10. I made J = 201 and ended at: Tue Sep 18 21:04:36 2018
. I took 757 nanosecs.

I am prio 20 thread and I started at: Tue Sep 18 21:04:36 2018
I am prio 20. I made J = 201 and ended at: Tue Sep 18 21:04:36 2018
. I took 752 nanosecs.

I am prio 10 thread and I started at: Tue Sep 18 21:04:36 2018
I am prio 10. I made J = 201 and ended at: Tue Sep 18 21:04:36 2018
. I took 746 nanosecs.

I am prio 30 thread and I started at: Tue Sep 18 21:04:37 2018
I am prio 30. I made J = 1 and ended at: Tue Sep 18 21:04:37 2018
. I took 176 nanosecs.

I am prio 10 thread and I started at: Tue Sep 18 21:04:37 2018
I am prio 10. I made J = 201 and ended at: Tue Sep 18 21:04:37 2018
. I took 741 nanosecs.

I am prio 20 thread and I started at: Tue Sep 18 21:04:37 2018
I am prio 20. I made J = 201 and ended at: Tue Sep 18 21:04:37 2018
. I took 608 nanosecs.

I am prio 10 thread and I started at: Tue Sep 18 21:04:37 2018
I am prio 10. I made J = 201 and ended at: Tue Sep 18 21:04:37 2018
. I took 623 nanosecs.

I am an Aperiodic Task and I started at: Tue Sep 18 21:04:37 2018
I am an Aperiodic Task and I ended at: Tue Sep 18 21:04:37 2018
. I took 104 nanosecs.

I am prio 10 thread and I started at: Tue Sep 18 21:04:37 2018
I am prio 10. I made J = 201 and ended at: Tue Sep 18 21:04:37 2018
. I took 607 nanosecs.

I am an Aperiodic Task and I started at: Tue Sep 18 21:04:37 2018
I am an Aperiodic Task and I ended at: Tue Sep 18 21:04:37 2018
. I took 399 nanosecs.

I am prio 10 thread and I started at: Tue Sep 18 21:04:37 2018
I am prio 10. I made J = 201 and ended at: Tue Sep 18 21:04:37 2018
```

Illustration 3: Log file screenshot