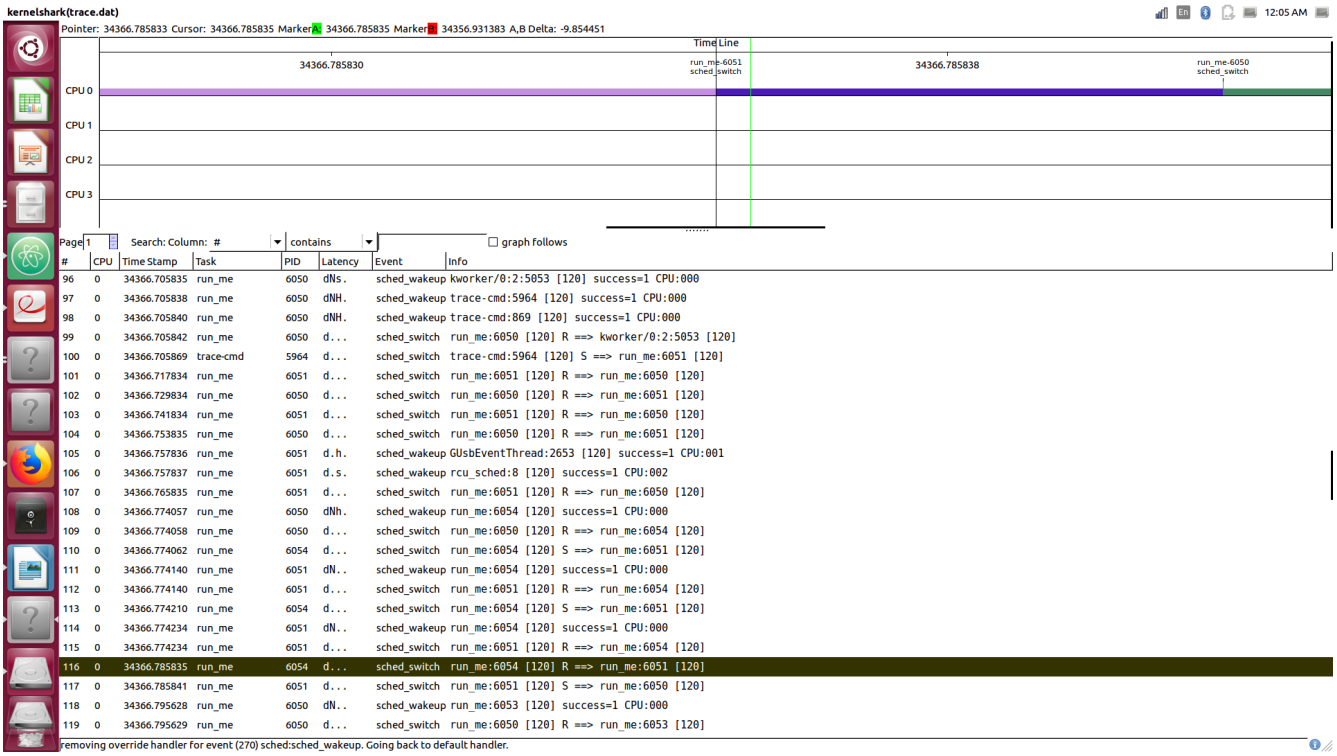
To demonstrate the priority inversion, we created a text file which has 4 threads with different priorities.
The threads are as follows
Periodic thread1 – priority 10
Periodic thread2 – priority 50
Aperiodic thread1 – priority 40
Aperiodic thread2 – priority 20

Based on the kernel-trace plot shown below(generated using Kernelshark and trace-cmd), we demonstrate priority inversion.
There are two periodic tasks in this example. The first task is PID 6051 running with the priority of 50. The second task is PID 6054 running with the priority of 10.



Ideally, without mutex locking, we would see 6051 pre-empting 6054 task, and running. However, we have mutex locks in both 6051 and 6054.

Both 6051 and 6054 need to acquire the same mutex lock. In the screenshot, we can see that on line 108,task 6054 got the lock and is now holding on to the lock. On line 112(Check screenshot), we can see that 6051 tries to take the lock, but doesn't succeed. So, 6051 has to wait for a little while longer, till 6054 releases the lock. Finally, on line 121, we can see that 6051 was able to acquire the CPU, after 6054 released the lock.

This is a classic case of priority inversion. A lower priority task(6054) keeps using the CPU even though a higher priority task is ready(6051). Priority inversion was possible in this case only because lower priority task locked the resource(mutex) and 6051 had to wait till 6054 was done processing