RESULTS

TASK MODEL
TOTAL PERIOD 10s ; TOTAL TASKS : 3

/* All periodic tasks*/

TASK1 : Priority 30 Period 1.75s <LM 3 LM 4 Compute Block : 0.25s UM4 UM3> (15042/15165)
TASK2 : Priority 20 Period 1.5s <LM 5 Compute Block : 1s UM 5> (15043/15166)
TASK3 : Priority 10 Period 10s < LM 3 Compute Block : 2s UM 3> (15044/15167)

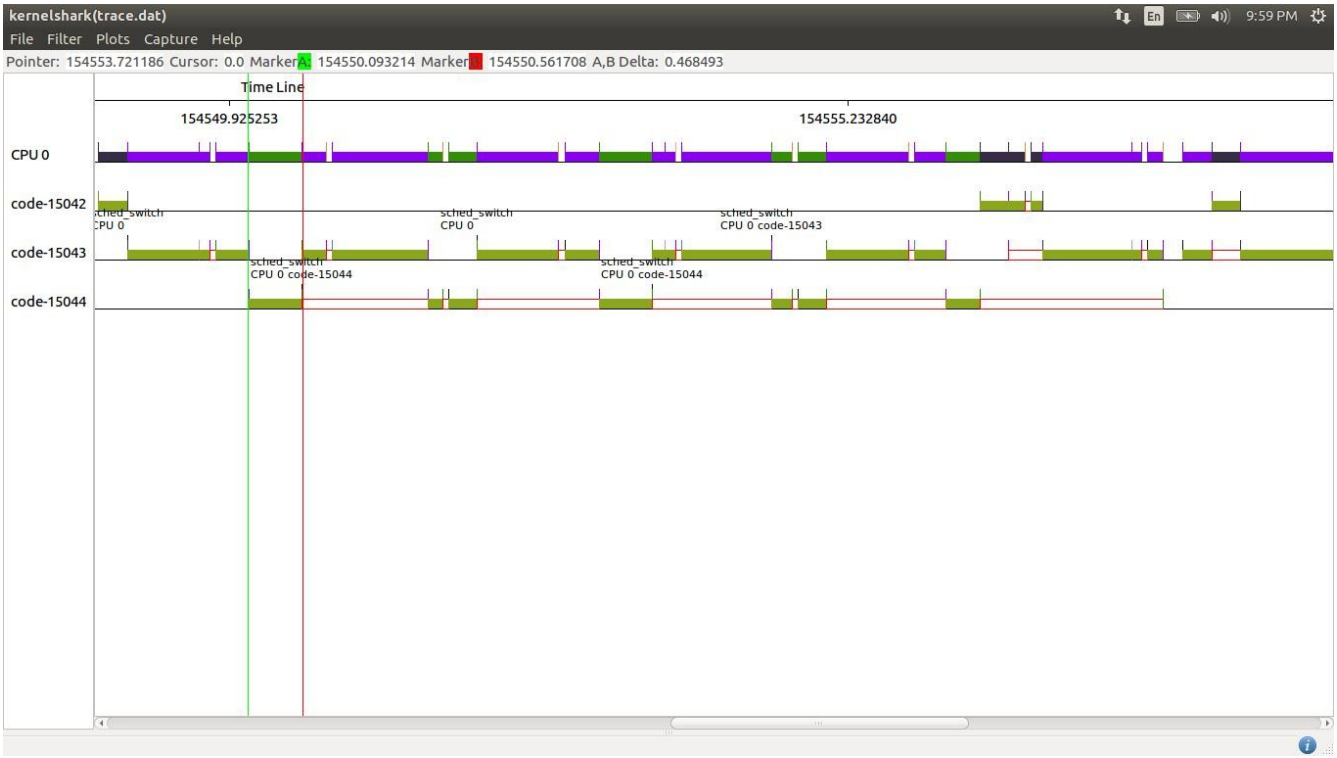

Figure1 is the kernelshark output of the task model with the input.txt(included in the folder) as input. **(PI_ENABLED = 0)**

In the beginning we see that the highest priority task comes in first executes the compute block and yields the CPU for the rest of it's period. Then the middle priority task executes and yields the CPU for the rest of it's period. Now the lowest priority task gets to run on the CPU which acquires a mutex lock and starts computing for 2s such that before it can unlock the mutex, middle priority task has already been woken up and preempts the lower priority task. Now when the highest priority task is woken up, it doesn't get to run because it can't acquire the lock and goes back to sleep. Due to this particular arrangement the highest priority task can't acquire the lock till the end of the period(until the lowest priority task unlocks it) but middle priority task which has a lower priority than highest priority task gets to run despite having no shared resources. Thus we see priority inversion here.

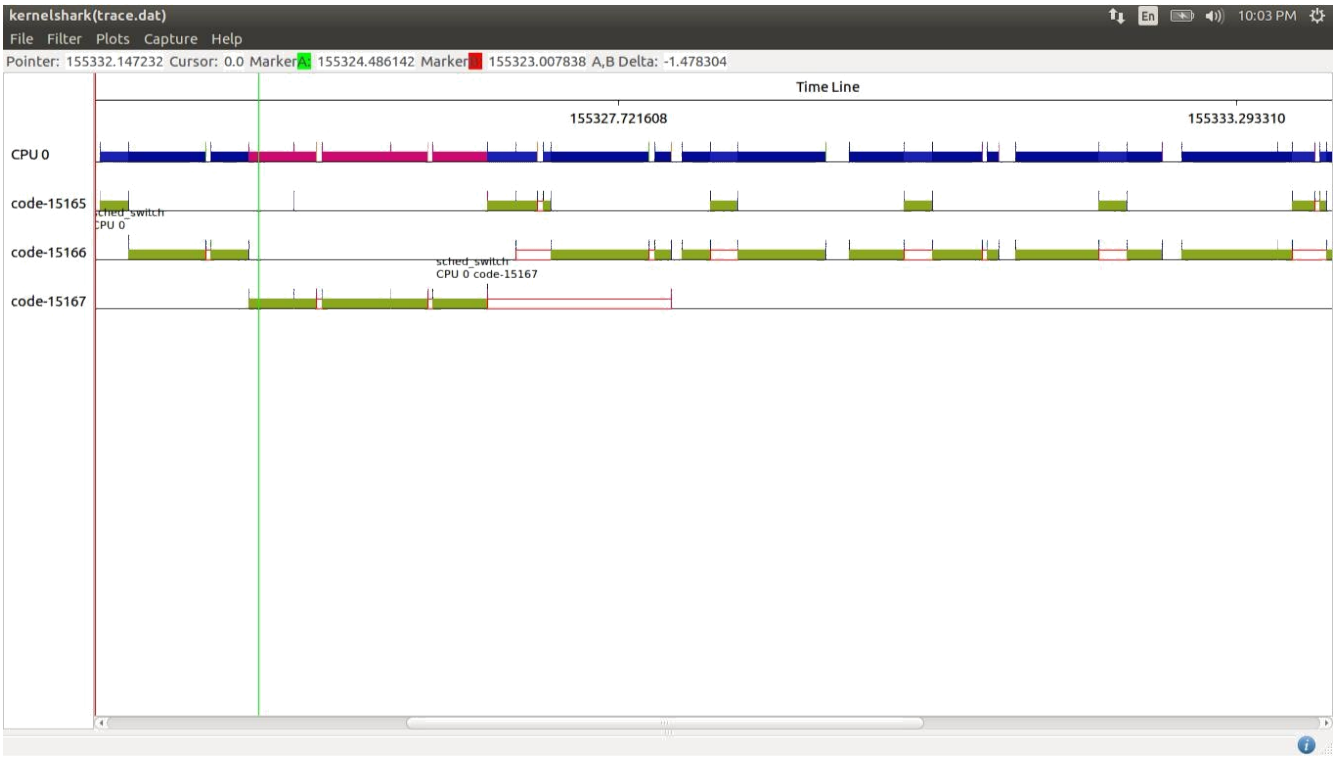

Figure 2 is the kernelshark output of the task model with the input.txt(included in the folder) as input. **(PI_ENABLED = 1)**

Here, we see that at the end of the period of the first task. When the first task is woken up the lowest priority task has the priority of the first task and thus until it yields the CPU, the middle priority task can't preempt it. After that the lowest priority task which had inherited the highest priority goes to sleep and the task2 and task1 share the CPU based on their priorities.