Scheduling Dual Criticality System in LITMUS-RT

Sethupandi Abishek
School of Computer Science and Engineering
Nanyang Technological University
Singapore
Email: abishek001@e.ntu.edu.sg

***Abstract*—This paper describes the implementation of dual criticality task system using Zero slack Rate monotonic Scheduler in LITMUS-RT. The mixed criticality systems can force tasks with different criticality levels. Based on an offline zero slack calculation algorithm, we assure the low criticality tasks can never stop the high criticality tasks (Criticality inversion problem). ZSRM also provides same level of utilization as RM, when all the tasks criticality levels are equal to its priorities and better if they are different.**

***Keywords*—Zero Slack, Litmus, Rate Monotonic, Linux**

I. Introduction

In real-time embedded systems such as automotive systems, there is a cost reduction trend by accommodating different functionality of different criticality into a shared hardware resources. When sharing the processor, if the lower criticality tasks using the processor for longer time, it could make the higher criticality tasks to miss its deadline. This is called critical inversion problem. The [1] proposes a scheduling policy called zero slack scheduling using an offline algorithm. This forms a protection that eliminates the impact on deadline due to critical inversion problem.

The paper is organized as follows. Section II summarizes the related work on zero slack rate monotonic scheduling. Section III and IV introduces the task model and the method of approach. Section V explains the implementation of ZSRM in Litmus-rt. Section VI shows our testing methods. We conclude with the future work in Section VII.

II. Related Work

D. De Niz et al present in [1] a scheme, all criticality levels have a guarantee on their execution time. The zero slack calculation algorithm does not impose more interference than required, which assures the scheduling guarantee in mixed criticality system. Their objective is to focus on deadlines and ensure that a high criticality task will not miss its deadline due to the allocation of more resources to a lower criticality task.

Their scheme identifies the last instant to block interference for a task τ_i, which the prevention mechanism can free required cycles for τ_i to complete its overloaded budget. This instant is called Zero slack instant. This mechanism maximizes the total schedulable utilization. It provides the same level of protection against the method of criticality as priority (CAPA). They have also introduced two metrics MC blocking and laxity utilization to measure the reduction of MC blocking for a given scheduling algorithm.

III. Design

To support both priorities and criticality levels of the task, The task τ_i has to be model as below,

$$\tau_i = (C_{LOW}, C_{HIGH}, T_i, D_i, \zeta, \text{ZS})$$

Where:
- C_{LOW} is a worst case execution time under non-overloaded condition.
- C_{HIGH} is the overloaded execution budget
- T_i is time period of the task
- D_i is Deadline of the task ($D_i = T_i$)
- ζ is the criticality of the task (0 - LOW criticality and 1 - HIGH Criticality)
- ZS is the zero slack instant calculated based on an offline algorithm proposed in [1]

IV. Methodology

A. LITMUS-RT

LITMUS-RT is an extension of Linux that supports real time multiprocessor scheduling policies. The ZSRM implementation is based on P-FP, partitioning fixed priority scheduling. The plugin assigns fixed priority to all tasks based on tasks relative period. It releases the respective task into ready queue. The scheduler picks the highest priority task and executes the job. When a new job is released, scheduler can preempt the current job and picks the highest priority job, which needs to be executed.

B. Zero Slack Enforcement

ZSRM plugin has two critical mode of operation. First, LOW criticality mode, It schedules all the task based on Rate monotonic scheduling policy with the budget of C_{LOW} for each task. Second, HIGH criticality mode, All the low criticality tasks are suspended and high criticality tasks are allowed to be executed with the budget of C_{HIGH}.

The zero slack timer monitors the job completion status at zero slack instant. Based on an offline zero slack calculation algorithm, Zero slack instant is known for each task. The absolute value of ZS instants are store in the datastructure. Timer is armed for minimum ZS instant as a callback time, which is equal to the sum of release time of the job and zero slack instant of the task. When the zero slack instant time for a job is met and the job has not completed, then the callback function changes the criticality mode from LOW to HIGH

C. Budget Enforcement

Budget Enforcement can be enabled in task model. The enforcement activates a budget timer which accurately police the slice budgets of each job in both LOW and HIGH criticality mode. In LOW Critic mode, budget timer is armed for C_{LOW} and for HIGH Critic mode as C_{HIGH}.

V. Implementation

This section explains implementation of ZSRM scheduler plugin in LITMUS-RT.

A. ZSRM Plugin

The sched_plugin structure have function pointers for basic functionality similar to P-FP plugin. It can have extra function pointers by adding the required elements in the structure declared in litmus/sched_plugin.h.

The zsrm plugin object is defined in litmus/sched_zsrm.c under the name of P-ZSRM. The get_global_criticality_t and set_global_criticality_t are the two new added function pointers in the sched_plugin structure to write and read the criticality mode.

B. Priority Assignement

When the tasks are added, they can be blocked for synchronous release using -w option in the rtspin task. These tasks are released synchronously using release_ts binary in liblitmus, a userspace library. Before releasing, All the tasks are assigned with new fixed priority values based on relative period using default bheap data structure.

The bheap structure, named rm_heap is declared inside the zsrm_domain_t structure. rm_heap is initialized in zsrm_domain_init() using bheap_init(). rm_ready_order, a comparator function based on relative period of the task, is defined in litmus/sched_plugin.h. It is used to build the bheap as a min bheap.

When a new task is added to the plugin, bheap_add function adds the task to rm_heap. When they are synchronously released, zsrm_task_wake_up assigns priority to all the tasks. bheap_take function extracts the minimum relative period node in rm_heap. The extracted task is assigned with the highest priority and the node is removed from rmheap. This above method is repeated until the rmheap is empty.

C. Zero Slack Instant Timer

The update_slack_enforcement_timer() function is called during scheduling from kernel/sched/litmus.c. The later function arms or cancel the timer. It can be armed only in low criticality mode. The Zero slack instant timers arming, canceling and call back functions are defined in litmus/zeroslack_timer.c.

The ZS timer is armed for every absolute value of ZS instant. The ZS instant values are received from the plugin, where the values are maintained in zs_heap data structure of zsrm domain. When a job releases, the respective tasks are added to zs_heap based on absolute ZS instant and the job completion status is updated to zero.

D. Release and Ready Queue

Release queue is defined inside rt_domain, which triggers an event for releasing jobs at the relative period. But the tasks are released to ready queue inside plugin with the help of zsrm_release_jobs() function. The later function also checks for the criticality mode of the task and the system mode to drop the low criticality tasks after in HIGH criticality mode.

Ready queue is declared in zsrm_ domain_t structure as fp_prio_queue, which is bitmask-indexed priority queue. Tasks are added to the bit mapped priority queue, where each index is a bheap. fpq_set, fpq_clear and fpq_find are the bit masking operations to set the index , to clear the index and to find the maximum priority. When a job is preempted, it can be moved to release queue or ready queue using requeue function, which is defined in plugin.

E. ZSRM Scheduler

ZSRM scheduler picks the high priority task and return as next task structure variable. When the criticality mode switched to HIGH, It handles two cases for zero slack enforcement. Case 1 - To drop both running and waiting, low criticality tasks in the ready queue. Case 2 - To notify the task / enforce the budget as C_{HIGH} for high criticality tasks. This can be achieved by assigning C_{HIGH} to tasks execution cost parameter in update_mode_to_userspace() function, which is defined in the plugin.

VI. Experimental Setup and Results

The ZSRM Scheduler is tested on odroid XU4 Platform. The platform runs on LITMUS-RT 2015.04 version which is based on kernel version 3.10.82+. The ZSRM scheduler is tested for single core processor in odroid platform as the other cores are disabled.

Lib-Litmus provides user space interface to interact with litmus schedulers. The scheduler plugin are tested with rtspin task sets, which acts as a simple cpu intensive tasks. The customized rtspin can handle parameters like execution time (high and low), relative period, criticality mode, deadline and zero slack instant of the tasks. The rtspin can switch between two different execution time like C_{LOW} and C_{HIGH}. Each rtspin monitors its own control page for memory mapped information transfer. The virtual memory is shared between the rtspin process and the kernel. Hence ZSRM scheduler can update the mode to user space tasks.

Consider the task sets in Table I and II for two different scenarios. Note that the priority and the criticality of the tasks in the tables are inversed. We tested the plugin with these two task sets.

A. ZS instant during job execution

When a second job of low criticality task is executing, the zero slack instant of HC task has occured in Fig 1. The mode has been switched to high, where the LC task is dropped and HC task is allowed to execute with the overloaded budget.

TABLE I. TWO-TASK EXAMPLE: ZS INSTANT DURING EXECUTION

Task	C_{LOW}	C_{HIGH}	T	D	Criticality	Priority	ZS
τ_0	2	2	5	5	1	0	3
τ_1	5	7	10	10	0	1	6

TABLE II. TWO-TASK EXAMPLE: ZS INSTANT BEFORE EXECUTION

Task	C_{LOW}	C_{HIGH}	T	D	Criticality	Priority	ZS
τ_0	2	2	5	5	1	0	3
τ_1	2	5	7	7	0	1	2

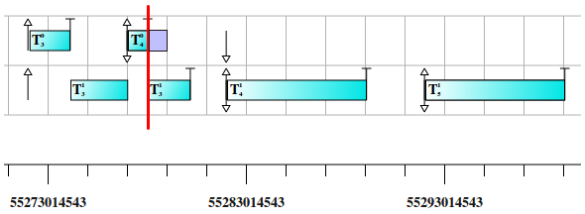

Fig. 1. Two-Task Example of ZS instant during execution of LC

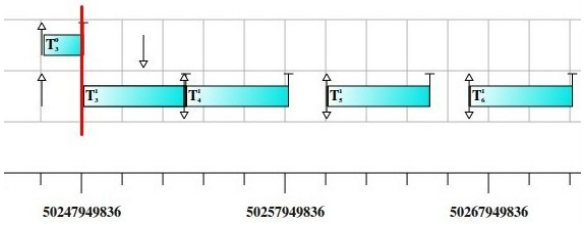

Fig. 2. Two-Task Example of ZS instant before HC execution

B. ZS instant before job execution

Here, The zero slack instant of HC task has occured before the job execution in Fig 2 and the HC task are allowed to execute with overloaded budget.

Thus ensuring all the high criticality tasks can be executed without missing deadline with overloaded budget during HIGH critic mode.

VII. CONCLUSION

In this paper, We discussed the implementation of ZSRM Scheduler in LITMUS-RT. The Scheduler is based on an offline zero slack instant calculation algorithm for Rate monotonic technique. Also it guarantee that the high criticality tasks are not stopped by lower criticality tasks in dual criticality system. The plugin is not supported for switching the mode from HIGH to LOW. It can be extended to support for multiprocessor using partitioned scheduling in LITMUS-RT.

ACKNOWLEDGMENT

I would like to express my sincere gratitude to our project guide Ramanathan Saravanan and Chandrashekaran Nair Bibin for the continuous support in this academic project. Their guidance helped me in all the time of course work.

REFERENCES

[1] D. De Niz, K. Lakshmanan, and R. R. Rajkumar, *On the scheduling of mixed-criticality real-time task sets*, In Proc. of 30th IEEE Real-Time Systems Symposium (RTSS), pages 291-300, 2009.