Fully Preemptive nxtOSEK Kernel with Preemption Threshold Scheduling

Motivation :

In real-time computing scenarios, we are concerned with logical as well as temporal accuracy of the results. While developing algorithms or performing schedulability analysis, preemption overheads are usually ignored even, though it's an operation that consumes finite amount of time. Preemption allows higher priority tasks to halt the lower priority tasks, though this need not be necessarily required to meet task deadlines. Our concern is not to get the results as soon as possible but soon enough to just meet the deadlines of all the tasks.

Preemption Threshold Scheduling (PTS) is an algorithm that tries to minimize preemptions by assigning two sets of priorities to each task. The nominal priority defines whether a task can preempt another task or not and the preemption-threshold is the effective priority with which a task runs during execution. When a task begins execution its priority is raised to preemption-threshold. Hence, only tasks with priority more than the preemption-threshold can preempt the running task. To assign the preemption-threshold, we initially assign it equal to the nominal priority. Starting with the task that has the least preemption-threshold we begin raising the priority till the point where the system is no more schedulable. We do this for all the tasks, thus creating mutually non-preemptive sets of tasks that bring down the number of preemptions while still meeting real time constraints

We chose the nxtOSEK kernel to demonstrate PTS implementation. However, the nxtOSEK kernel does not support preemption as expected. The task scheduling in the nxtOSEK kernel is governed by a 1ms ISR routine. When a higher priority task preempts a lower priority task, it has been observed that this 1ms ISR(and hence the task scheduler) is suppressed for the duration of execution of the higher priority task. This causes an equivalent shift in the release times of all the subsequent tasks and the entire timing behaviour of the nxtOSEK kernel gets skewed.

We believe the nxtOSEK kernel has been purposefully designed as such to prevent successive instances of preemption i.e. the kernel does not support a case where after one preemption event, while the preempting task is yet to finish it's execution, it is preempted in turn by an even higher priority task. This is because, during preemption, the context of the lower priority task is saved on the system stack, and hence allowing for successive preemptions could potentially corrupt the context of lower priority tasks. Such a behavior comes at the cost of skewing the timing reference of the kernel as the 1ms ISR responsible for scheduling is disabled while the higher priority task is running after preemption.

We shall attempt to resolve this by allocating a protected area for saving context for each task so that successive preemptions can be allowed.

Deliverables:

Primary: Debug, resolve and validate preemption in nxtOSEK

Secondary: Implement PTS and validate it on nxtOSEK.

Saransh Gupta & Jimit Doshi

Milestones:

3/27/2014 - Understand the rationale behind existing implementation and consolidate all the efforts so far to resolve the issue with preemption.
4/4/2014 - Modify the kernel implementation to support fully preemptive scenarios.
4/15/2014 - Test & validate the design.
4/21/2014 - Integrate PTS algorithm with existing kernel.
4/25/2014 - Validate PTS.
4/27/2014 - Presentation
5/1/2014 - Final Project Report

Project Website:

http://www4.ncsu.edu/~sgupta20/pts.html

References:

1. Using Preemption-Threshold Scheduling to Cut Overhead While Meeting Deadlines, by Dr. Alex Dean
2. Embedded Systems Design Magazine, March, 2011, Feature Article: "Lower the Overhead in RTOS Scheduling," by Professor Alexander Dean, Ph.D.
3. nxtOSEK Homepage