Swift mode changes in memory constrained real-time systems

Mike Holenderski

CANTATA workshop, April 2009

Context

- Memory constrained platform
  - Not all applications can operate simultaneously
    - Reduce number of applications
    - Allow applications to trade quality for memory
      - Scalable applications

Context

- Scalable application
  - **Application** is a set of tasks
  - **Task** specifies part of the application workload
  - **Scalable** application
    - Can adapt its workload to available resources
    - Operates in one of predefined modes
  - **Mode** specifies task parameters (memory requirements)
- During runtime system may decide to change mode, i.e. reallocate the memory between tasks
- Quality Manager Component (QMC)
  - Initiates and coordinates mode changes

A real-time application is modeled by a set of tasks, where each task specifies part of the application workload.

Scalable applications are those which can adapt their workload to the
available resources. In this presentation we consider scalable applications which can operate in one of several predefined modes.

A mode specifies the task parameters (i.e. the resource requirements) of the application task set.

Problem

- Mode change latency:
  - Time between a *mode change request* and the time when all tasks have changed to their target mode
- Problem: bound the latency of a mode change

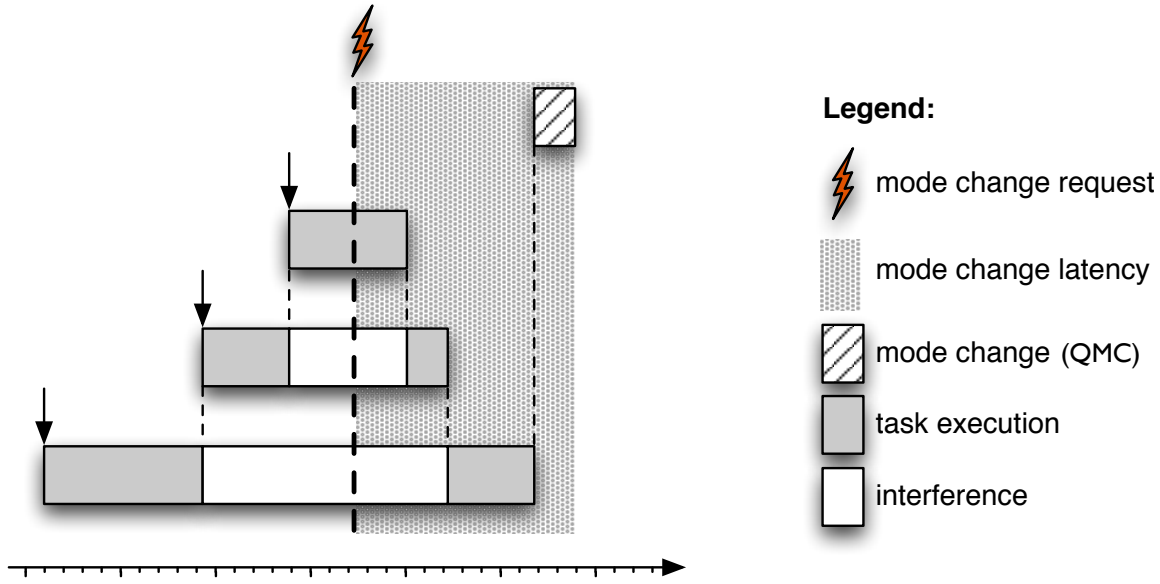
Overview of the design space

- Memory Management Unit (MMU) is available
  - Memory can be easily reallocated between tasks (paging)
- No MMU, but data can be moved (at additional cost)
  - Platform does not have an MMU, or MMU is present but not used (too expensive)
  - e.g. DMA support for moving data around in memory
  - Moving data in memory incurs some overhead
- No MMU, and data cannot be moved in memory
  - Moving data in memory incurs too high overhead

We have addressed all three classes of systems. In this presentation we show a part common to all: introducing non-preemption to improve the latency bound (compared to existing preemptive scheduling).

Existing approach (literature)

- Fixed Priority Preemptive Scheduling (FPPS)



This diagram shows tasks **involved in the mode change**. This set of tasks depends on the class of the system (previous slide). For example, if there is no support for cheap data copying within memory (e.g. no MMU) then the set of tasks involved in the mode change will also include those tasks, which are not required to change their memory requirements, but which have to be reallocated to accommodate the mode change in other tasks (due to fragmentation).

Upon a mode change request all ready tasks **involved in the mode change** have to complete before the mode can be changed.

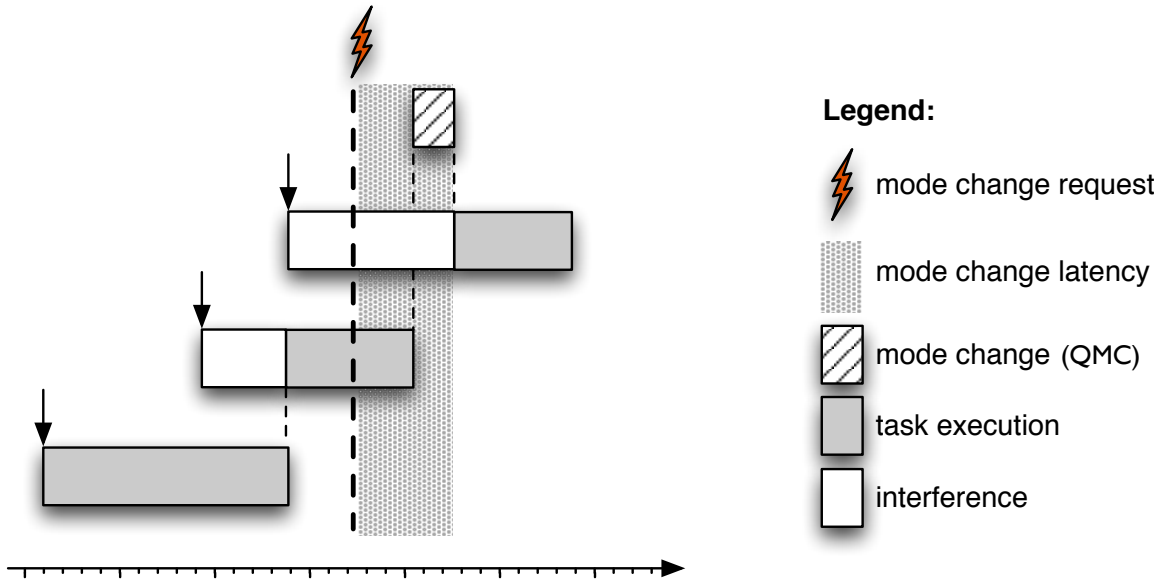
Requires communication between these tasks and the QMC and complicated protocol to identify when tasks have completed. (E.g. if the priority of these tasks is raised above others, then, in case of resource sharing, the resource access protocol may need to be adapted).

Our approach

- Introduce non-preemption (or deferred preemption)
- Assign QMC the highest priority
- Let the QMC execute the mode changes on behalf of the tasks
  - Each task implements a method responsible for adapting the task to the target mode (e.g. reducing the memory requirements)
  - This method is called by the QMC

Our approach

- Fixed Priority Non-preemptive Scheduling (FPNS)



8

Upon a mode change request only the currently running task has to complete before the mode can be changed.

The non-preemption gets rid of the need for tasks communicating when they have finished.

Because FPNS may reduce the schedulability of the higher priority tasks, hence apply FPDS instead of FPNS, with preemption points chosen in such a way that the latency bound of a mode change is minimized, while the task set is kept schedulable.

In case of deferred preemption, assume that mode changes can be performed at preemption points.

Some math

- FPPS $\mathcal{L}^P(\gamma) = \sum_{\tau_i \in \gamma} C_i + |\gamma| C_{sys}$
- FPNS $\mathcal{L}^D(\gamma) = \max_{\tau_i \in \gamma} C_i + |\gamma| C_{sys}$

γ	set of tasks involved in the mode change
$\mathcal{L}(\gamma)$	latency of a system mode change involving tasks in γ
C_i	worst-case computation time if task τ_i
C_{sys}	worst-case system overhead for changing a single component mode (e.g. memory allocation)

The difference is in the sum vs. max.

In case of FPDS, the equations become a little more complicated, but the difference remains in sum vs. max.

Questions?