Priority Assignment in Fixed Priority Pre-emptive Systems with Varying Context Switch Costs

Robert I. Davis\textsuperscript{1}, Sebastian Altmeyer\textsuperscript{2}, Alan Burns\textsuperscript{1}

\textsuperscript{1}Real-Time Systems Research Group, University of York, UK
\textsuperscript{2}University of Amsterdam (UvA), Amsterdam, Netherlands
Motivation:
Temporal and Spatial Separation

- Separation is vitally important
  - Safety standards (IEC61508, DO-178C, ISO26262) require that either all applications are developed to the standard required for the highest criticality application, or that independence between different applications is demonstrated in both spatial and temporal domains.

- Process and Thread model
  - Each process has a separate memory address space
  - Threads within a process share the same address space

- Enables spatial and temporal separation
  - By mapping all tasks from a given application to a distinct process (one process per application)
  - Or by mapping all tasks of a given criticality level to a distinct process (one process per criticality level)

Processes and Threads

Key point: Two very different context switch costs
System Model

- Single processor
- Fixed priority pre-emptive scheduling (FPPS)
- Sporadic tasks
  - Each task $\tau_i$
  - $T_i$ – Period or minimum inter-arrival time (sporadic behaviour)
  - $D_i$ – Constrained relative deadline
  - $C_i$ – worst-case execution time

- Additionally
  - Each task is mapped to an address space $A_i$ (and process)
  - When one task $\tau_i$ pre-empts another task $\tau_j$
    - same address space ($A_i = A_j$) implies a small context switch cost $C_s$
    - change in address space ($A_i \neq A_j$) implies a large context switch cost $C_c$
    (Here costs are for switching from and later back to the pre-empted task)
Response Time Analysis for FPPS: Simple Analysis

- Method

  - Use large context switch cost $C_C$ for every pre-emption
  - Equivalent to subsuming context switch times into WCET bounds
  - Response time for task $\tau_i$

  $$R_i = C_i + C_C + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil (C_j + C_C)$$

  - Fixed point iteration (converges or ends when value exceeds $D_i$)
Response Time Analysis for FPPS: Simple Analysis

- Example:
  - Three tasks with parameters \((C_i, D_i, T_i, L_i, A_i)\)
    \[
    \tau_A = (10, 50, 100, LO, A^L) \\
    \tau_B = (10, 100, 200, HI, A^H) \\
    \tau_C = (200, 265, 300, LO, A^L)
    \]
  - Further \(C^C = 5\) and \(C^S = 0\)
  - Deadline Monotonic Priority Order (DMPO) is optimal
  - With priority order \(\{A, B, C\}\) then \(R_C = 280\) hence task set is not schedulable

- Part of schedule illustrating context switch costs

![Diagram showing context switch costs]

- HI-criticality execution
- LO-criticality execution
- Large context switch cost (every time)
Response Time Analysis for FPPS: Refined Analysis

- **Method**
  - Consider the set of tasks $\text{aff}(i, j) = \text{hep}(i) \cap \text{lp}(j)$ that can be affected by pre-emption by task $\tau_j$ during the response time of task $\tau_i$

    \[
    \text{Priority} \quad \text{aff}(i, j) \quad \begin{cases} 
    \tau_j \\
    \tau_x \\
    \tau_i 
    \end{cases}
    \]

  - Only get a large context switch cost for pre-emption by task $\tau_j$ if there is some task $\tau_h$ that can be pre-empted by task $\tau_j$ during the response time of task $\tau_i$ that belongs to a different process and hence different address space

    \[
    \gamma_{i,j} = \begin{cases} 
    C^C & \text{if } \exists h \in \text{aff}(i, j) | A_h \neq A_j \\
    C^S & \text{otherwise}
    \end{cases}
    \]

    \[
    R_i = C_i + C^C + \sum_{\forall j \in \text{hp}(i)} \left[ \frac{R_i}{T_j} \right] (C_j + \gamma_{i,j})
    \]
Response Time Analysis for FPPS: Refined Analysis

- Example:
  - Three tasks with parameters \((C_i, D_i, T_i, L_i, A_i)\)
    \[
    \tau_A = (10, 50, 100, LO, A^L) \\
    \tau_B = (10, 100, 200, HI, A^H) \\
    \tau_C = (200, 265, 300, LO, A^L)
    \]
  - Further \(C^C = 5\) and \(C^S = 0\)
  - Deadline Monotonic Priority Order (DMPO) is not optimal
  - With priority order \(\{A, B, C\}\) then \(R_C = 280\) hence task set is not schedulable
  - With priority order \(\{B, A, C\}\), then \(R_C = 265\) and task set is schedulable

- Part of schedule illustrating context switch costs

Shared process and address space implies small context switch cost
Response Time Analysis for FPPS: Multiset Analysis

- See the RTAS 2018 paper for details of the multiset analysis
  - Multiset analysis dominates the refined analysis

Open Problem: Efficient Optimal Priority Assignment

- How to efficiently obtain an optimal priority assignment* with respect to the refined analysis? (and with respect to the multiset analysis?)

*An optimal priority assignment is one that is schedulable whenever there exists a schedulable priority assignment for the system.
Audsley’s algorithm: Optimal Priority Assignment (OPA)

for each priority level i, lowest first {
    for each unassigned task \( \tau \) {
        if \( \tau \) is schedulable at priority i
            assuming that all unassigned tasks are
            at higher priorities {
                assign task \( \tau \) to priority level i
                break (exit for loop)
            }
    }
    if no tasks are schedulable at priority i {
        return unschedulable
    }
}
return schedulable

\( n(n+1)/2 \) schedulability tests rather than \( n! \)
by exhaustively exploring all possible orderings

(e.g. for \( n=15 \), 120 schedulability tests compared to 1307674368000)
OPA algorithm applicability

OPA algorithm provides optimal priority assignment with respect to any schedulability test $S$ for fixed priority scheduling provided that three conditions are met...

**Condition 1:** Schedulability of a task may, according to the test, be dependent on the set of higher priority tasks, but not on their relative priority ordering.

**Condition 2:** Schedulability of a task may, according to the test, be dependent on the set of lower priority tasks, but not on their relative priority ordering.

**Condition 3:** When the priorities of any two tasks of adjacent priority are swapped, the task being assigned a higher priority cannot become unschedulable according to the test, if it was previously deemed schedulable at the lower priority.

Priority assignment toolbox: Techniques to explore #1

- Task swapping
  - Idea is to establish rules under which schedulability continues to hold when we swap two specific tasks in the priority order (This is the basis of many proofs of optimal priority orderings)
  - Can then use those rules to transform any schedulable ordering into another one with those tasks in a particular order without loss of schedulability
  - This might provide additional information / properties that hold for an optimal priority ordering which can then be used to reduce the complexity of finding it

- Hints and tips
  - If we swap two tasks from the same process under what circumstances would they both remain schedulable?

Priority assignment toolbox: Techniques to explore #2

- Results from research into Robust Priority Assignment
  - Prior work on Robust Priority Assignment has shown that Deadline Monotonic is the optimal priority order for tasks subject to an additional interference function
  - Additional interference function is very general - only has to be monotonically non-decreasing with respect to lower priority levels and increasing intervals over which interference is considered

- Hints and tips
  - Perhaps it would be useful to consider a sub-set of tasks belonging to a specific process and regard all other tasks as just an additional interference function – we might then be able to show that Deadline Monotonic partial order is optimal for the sub-set of tasks in each process under the refined analysis?

Priority assignment toolbox: Techniques to explore #3

- **Sufficient test**
  - If a task is schedulable at the lowest (unassigned) priority assuming a simple analysis (with all context switch costs assumed to be large) then it MUST be schedulable at that level with the refined and multiset analysis irrespective of the order of higher priority tasks.

- **Necessary test**
  - If a task is unschedulable at the lowest (unassigned) priority assuming a simple analysis (with all context switch costs assumed to be small) then it CANNOT be schedulable at that level (with the set of higher priority tasks unchanged) under refined or multiset analysis irrespective of the order of higher priority tasks.

- **Hints and tips**
  - Can these tests help us to build an optimal priority ordering for the refined analysis?

Open Problems

- How to efficiently obtain an optimal priority assignment for the refined analysis? (and for the multiset analysis?)
  - For two processes?
  - For multiple processes?

- How best to schedule tasks when there are two different context switch costs (process-level and thread-level)?
  - Fully pre-emptive scheduling has the disadvantage of a large number of context switches
  - What about using non-preemptive scheduling or limited preemption scheduling?