Optimal Fixed Priority Scheduling with Deferred Pre-emption

Outline

- The schedulability of systems using fixed priority preemptive scheduling can be improved by the use of non-preemptive regions at the end of each task's execution (an approach referred to as deferred pre-emption).

- Choosing the appropriate length for the final non-preemptive region of each task is a trade-off between:
  - Improving the worst-case response time of the task
    AND
  - Increasing the amount of blocking imposed on higher priority tasks.

- This paper presents an optimal algorithm for determining **2 things**:
  - The priority ordering of tasks
  - The lengths of final non-preemptive regions of these tasks.

Claim

This algorithm is optimal for fixed priority deferred pre-emptive scheduling (FPDS), because it guarantees to find a schedulable combination of priority ordering (PA) and Final Non-preemptive Region (FNR) lengths if such a schedulable combination exists.

Proposed Algorithm

- FNR-PA: Final Non-preemptive Region & Priority Assignment
- This algorithms is optimal in the sense that it is able to find a schedulable solution for any taskset that is feasible under FPDS
- The algorithm relies on finding the MINIMUM final non-preemptive region length for each task that ensures its schedulability.

Background Research

- Add some more
- **2** different models of fixed priority scheduling with deferred pre-emption have been developed in the literature
  - *Fixed Model:*
    - The location of each non-pre-emptive region is statically determined prior to execution.
    - Pre-emption is only permitted at pre-defined locations in the code of each task

    - *Floating Model:*
        - An upper bound is given on the length of the longest non-pre-emptive region of each task.
        - The location of each non-pre-emptive region is not known a priori and may vary at run-time.
- In 2011 Bertogna derived a method of *computing the optimal length of the final non-pre-emptive region* of each task in order to maximize schedulability under FPDS *for a given priority assignment*

Assumptions:

- A form of fixed priority scheduling with deferred pre-emption where each task has a final non-preemptive region.
- A discrete time model is assumed, where all task parameters are assumed to be positive integers.
- Any resource access occurring within the final non-pre-emptive region of a task is properly nested, i.e. wholly included within that region, avoiding deadlocks.
- The arrival times of the tasks are independent and unknown a-priori. Hence the tasks may share a common release time.
- Each task is released (becomes ready to execute), as soon as it arrives. The tasks do not voluntarily suspend themselves.
- Due to the integer time model considered in this paper, the discrete time granularity is **1** time unit.

System Model, Terminology & Notation

- Consider the fixed priority scheduling of a set of sporadic tasks (or taskset) on a single processor.
- Each taskset comprises a static set of n tasks ($\mathbf{t_1}$ .. $\mathbf{t_n}$), where "**n**" is a positive integer.
- "i" denotes priority (t_1 has the highest ... t_n the lowest).
- A discrete time model is assumed, where all task parameters are positive integers.
- ***lp(i)*** : Set of tasks with priorities < "**i**"
- ***lep(i)*** : Set of tasks with priorities ≤"**i**"
- ***hp(i)*** : Set of tasks with priorities >"**i**"
- ***hep(i)*** : Set of tasks with priorities ≥"**i**"
- Worst case execution time of task $\mathbf{t_i}$ = $\mathbf{C_i}$
- Worst case response time of task $\mathbf{t_i}$ = $\mathbf{R_i}$
- Task period ($\mathbf{T_i}$)
- Relative Deadline ($\mathbf{D_i}$)
- Utilisation of task "$\mathbf{t_i}$" is $\mathbf{U_i = C_i / T_i}$
- Each task is assumed to have a final non-preemptive region of length $\mathbf{F_i}$ in the range [1, C_i].
- The tasks may make mutually exclusive access to a shared resource - using some Shared Resource Policy (SRP).
- $\mathbf{B_{li}}$ denotes - the longest time a task $\mathbf{t_l}$ may continue executing while holding a resource needed by another task of priority "i". [$\boldsymbol{t_l}$ ∈ ***lp(i)***]
- Therefore $\mathbf{B_{li}}$ = longest time a task $\mathbf{t_l}$ can execute at a priority "**i**" or higher.

- A taskset is schedulable iff , for all "**i**", $\mathbf{R_i \leq D_i}$.

- Under FPDS, at any given time the highest priority ready task is selected for execution by the processor
- Both the *final non-pre-emptive regions* and *resource accesses* according to SRP are implemented by manipulating task priorities.
- Thus a task executing a final non-pre-emptive region has the highest priority and so will not be pre-empted.

Types of Tasksets

- *Implicit-deadline* : A taskset where for all tasks Di = Ti (deadline = period)
- *Constraint-deadline* : A taskset where for all tasks Di ≤ Ti (deadline ≤ period)
- *Arbitrary-deadline*: A taskset where for all tasks deadlines are independent of their periods.

Definitions

- *Schedulability* : A taskset is said to be schedulable with respect to some scheduling algorithm, if all valid sequences of jobs that may be generated by the taskset can be scheduled by the algorithm without any missed deadlines.

- *Optimality* : A priority assignment policy P is said to be optimal with respect to some class of tasksets (e.g. arbitrary-deadline), and some type of fixed priority scheduling algorithm (e.g. FPPS, FPNS, or FPDS) - if there are no tasksets in the class that are schedulable under the scheduling algorithm using any other priority ordering policy, that are not also schedulable using the priority assignment determined by policy P.

- A fixed priority scheduling algorithm A is said to *dominate* another fixed priority scheduling algorithm B if there are tasksets that can be scheduled under algorithm A, but cannot be scheduled under algorithm B, and all of the tasksets that are schedulable under algorithm B are also schedulable under algorithm A.
- If there are tasksets that are schedulable under algorithm A, but not under algorithm B, and vice-versa, then the two algorithms are said to be *incomparable*.
- If both algorithms can schedule precisely the same tasksets then they are said to be *equivalent*.

- *Sustainability* of a *Scheduling Algorithm*:
  - A scheduling algorithm is said to be *sustainable with respect to a system model*, if and only if schedulability of any taskset compliant with the model implies schedulability of the same taskset modified by:
    1. decreasing execution times
    2. increasing periods or inter-arrival times, and
    3. increasing deadlines

- *Sustainability* of a *Schedulabiltiy Test*: A schedulability test is referred to as sustainable if the above changes (1,2,3) cannot result in a taskset that was previously deemed schedulable by the test

becoming unschedulable.

Summarising Prior Work:

- Schedulability analysis for fixed priority scheduling with deferred pre-emption for sporadic tasksets with arbitrary deadlines

Priority level-i active period

- A continuous period of time [t1,t2) during which tasks, of priority "**i**" or higher, that were released at the start of the active period at t1 , or during the active period but strictly before its end at t2 , are either executing or ready to execute.

- Critical Instance (for a task t_i)

- Occurs when $\mathbf{t_i}$ is released simultaneously with all higher priority tasks, and subsequent releases of task $\mathbf{t_i}$ and higher priority tasks occur after the minimum permitted time intervals.
- Here Ť refers to - the minimum possible amount of time prior to this simultaneous release, a lower priority task tk enters its final non-premptive region or begins accessing a resource shared with a task of priority "**i**" or higher.

- Previous work on FPDS established that, the longest response time of a task $\mathbf{t_i}$ occurs for some job of that task within the *priority level-i active period* starting at *-Critical Instant*.
- The worst-case length of a *priority level-i active period* $\mathbf{A_i}$ is given by the minimum solution to the following fixed point iteration.

$$A_i^{m+1} = B_i + \sum_{\forall j \in hep(i)} \left\lceil \frac{A_i^m}{T_j} \right\rceil C_j \qquad (1)$$

- Iteration starts with A^0_i = Ci, and ends with $\mathbf{Ai^{m+1} = Ai^m}$

- $\mathbf{B_i}$ is the longest time that the task $\mathbf{t_i}$ can be blocked from executing by the lower priority tasks. given as below:

$$B_i = \max(B_i^{RES}, B_i^{FNR}) \qquad (2)$$

$$B_i^{RES} = \max_{\forall l \in lp(i)} (B_l^i - 1), \quad B_i^{FNR} = \max_{\forall l \in lp(i)} (F_l - 1)$$

- $\mathbf{Bi^{RES}}$ and $\mathbf{Bi^{FNR}}$ are the blocking factors at priority "**i**" due to resource locking and final non-preemptive regions respectively.

- The number of jobs **G_i** of task "**i**" in the *priority level-i active period* is given by:

$$G_i = \left\lceil \frac{A_i}{T_i} \right\rceil \qquad (3)$$

- The start time **$W_{i,g}^{NP}$** of the final non-preemptive region of job "**g**" of task "**i**" measured with respect to the start of the Ť-critical instant, and is given by the minimum solution to the following fixed point iteration:

$$w_{i,g}^{m+1} = B_i + (g+1)C_i - F_i + \sum_{\forall j \in hp(i)} \left(\left\lfloor \frac{w_{i,q}^{m}}{T_j} \right\rfloor + 1 \right) C_j \qquad (4)$$

Iteration starts with an initial value $w_{i,g}^{0}$ guaranteed to be no greater than $W_{i,g}^{NP}$, typically $w_{i,g}^{0} = B_i + (g+1)C_i - F_i$, and ends when either $w_{i,g}^{m+1} = w_{i,g}^{m}$ in which case $W_{i,g}^{NP} = w_{i,g}^{m+1}$, or when: $w_{i,g}^{m+1} + F_i - gT_i > D_i$ in which case job g, and hence task τ_i is unschedulable.

- The worst case response time of of task **t_i** is given by:

$$R_i = \max_{\forall g=0,1,2...G_i-1} (W_{i,g}^{NP} + F_i - gT_i) \qquad (5)$$

- Task "**t_i**" is schedulable provided that **R_i ή D_i**

Corollary 1: *Sustainability of task schedulability with respect to an increase in the length of its final non-preemptive region*.

- Based on the equations (4) and (5) the length of the final non-preemptive region of a task (**F_i**) cannot result in that task becoming unschedulable if it was previously schedulable.
    - Reason is if **F_i** increases then the iterative solution in (4) will reduce the value of **$W_{i,g}^{NP}$** .. making the equation in (5) balanced.

Example explaining the FPDS using a Taskset.

Problem Descriptions

---

1. *Final Non-pre-emptive Region length Problem (**FNR Problem**):* For a given taskset complying with the

1. *Final Non-pre-emptive Region length Problem (**FNR** Problem):* For a given taskset complying with the task model described above and a given priority ordering X, find a value for the length of the final non-pre-emptive region of each task such that the taskset is schedulable under FPDS.
2. *Final Non-pre-emptive Region Length and Priority Assignment Problem (**FNR-PA** Problem):* For a given taskset complying with the task model described above, find both
   (i) a priority assignment, and
   (ii) a value for the length of the final non-pre-emptive region of each task that makes the taskset schedulable under FPDS.

Corollary 2:
A task that is schedulable at *priority level k* with a blocking factor *B(k)* due to a set of lower priority tasks *lp(k)* remains schedulable when the blocking factor is reduced and the sets *lp(k)* and *hp(k)* of lower and higher priority tasks remain unchanged.

Corollary 3:
The schedulability under FPDS of a task at *priority k* with a final non-pre-emptive region of length *F(k)* depends on the set of higher priority tasks *hp(k)* , but is independent of the relative priority ordering of those tasks.

Corollary 4:
The schedulability under FPDS of a task at priority k with a final non-pre-emptive region of length F(k) depends on the set *lp(k)* of lower priority tasks in respect of the blocking factor; however, there is no dependency on the relative priority ordering of the lower priority tasks.

Corollary 5:
- For a given set of higher priority tasks *hp(k)*, the minimum length *F(k)* of the final non-preemptive region of task $\mathbf{t_k}$ is a monotonically nondecreasing function of the blocking factor *B(k)*.
- Stated otherwise, a larger blocking factor at
cannot result in a smaller minimum length for the final non-pre-emptive region of the task at that priority level.

Corollary 6:
For a given taskset and fixed priority ordering X, that is schedulable under FPDS with some set of final non-pre-emptive region lengths, FNR Algorithm minimises the final non-pre-emptive region length of every task, and hence minimises the blocking factor at every priority level.

FNR Algorithm:

Algorithm

```
for each priority level k, lowest first
{
    - determine the smallest value for the final non-pre-emptive region length F(k)
      such that the task at priority k is schedulable.
    - Set the length of the final non-pre-emptive region of the task to this value.
```

```
}
```

Analytical method to find the FNR length

FNR-PA Algorithm:

Algorithm

```
for each priority level k, lowest first
{
    for each unassigned task
    {
        - determine the smallest value for the final non-pre-emptive region length F(k)
          such that task 't' is schedulable at priority k, assuming all other unassigned
          tasks have higher priorities.
        - Record as task 'Z' the unassigned task with the minimum value for the length
          of its final non-pre-emptive region F(k).
    }
    if no tasks are schedulable at priority k
    {
        return unschedulable
    }
    else
    {
        - assign priority k to task 'Z'
        - use the value of F(k) as the length of its final non-pre-emptive region.
    }
}
return schedulable
```

Explanation (How it works)

Backup:

Theorem 1:
The FNR algorithm is optimal for the FNR problem (Problem 1)

Theorem 2:
The Final Non-pre-emptive Region & Priority Assignment (FNR-PA) algorithm is optimal for the FNR-PA problem (Problem 2)

Theorem 3:
For any taskset where there exists a priority ordering and a set of final non-pre-emptive region lengths that is schedulable under FPDS, the FNR-PA algorithm results in a blocking factor FNR Bi from final non-preemptive regions at every priority level i that is no larger than that obtained with any other schedulable

priority and final non-pre-emptive region length assignment.