FKP: a constant-time algorithm to schedule hard real-time sporadic tasks

CISTER Periodic Seminar

Damien Masson[1]
Geoffrey Nelissen[2]

[1]Université Paris-Est, ESIEE Paris, Laboratoire d'informatique Gaspard-Monge (LIGM) UMR CNRS 8049

[2]CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto

Tuesday, June 30th 2015

Fixed-K Priority Scheduler

Fixed-K Priority Scheduler

Task Model

A sporadic task τ_i produces an infinite sequence of jobs and is defined by:

- its first release time instant: r_i
- its worst case execution time (WCET): C_i
- the minimal inter arrival time between two jobs: T_i
- its relative deadline: D_i

Scheduling problem

provide an algorithm to choose amongst the tasks at runtime. The schedule produced have to be deterministic enough to permits to answer the question: are the deadline respected in the worst case or not ?

Fixed Priority

Each task is assign a unique priority. The scheduler chooses the highest priority task amongst the ready ones.

- Deadline Monotonic (DM) is an optimal priority assignment for this class of scheduler, iff $\forall i, D_i \leq T_i$
- The processor utilization bound is approximately 70% for preemptive scheduler in the general case
- This kind of scheduler is implemented in constant time in all modern operating systems

Dynamic Priority

The priority of each job can change at any instant.

- Earliest Deadline First (EDF) is an optimal algorithm for this class of scheduler (in fact EDF belongs to the subclass of job-level fixed priority)
- The processor utilization bound is approximately 100% for preemptive scheduler
- EDF can be cleverly implemented (very recent work shows how) but the complexity is still not constant because one hardware timer is needed for each task, and these timers need to be sorted.

How far Fixed priority and Dynamic Priority assignment classes are ?

The Dual priority scheduler

This scheduler considers a set of n independent periodic tasks with implicit deadlines $\{\tau_1, \tau_2, ..., \tau_i, ..., \tau_n\}$ with $\tau_i = (r_i, C_i, T_i, S_i, P_i^1, P_i^2)$

- (r_i, C_i, T_i) are the usual parameters from the Liu&Layland model,
- S_i is the relative intermediate deadline where the task priority changes,
- P_i^1 and P_i^2 are respectively the priority before the intermediate deadline and after.

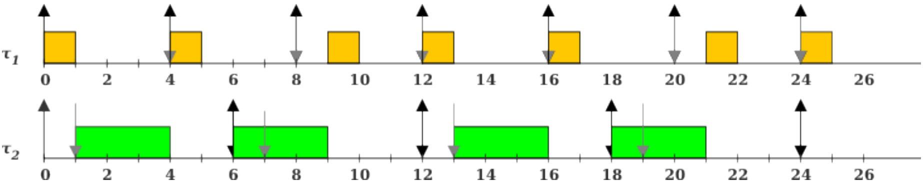


Figure: Example with $P_1^1 < P_2^1 < P_1^2 < P_2^2$

Conjectures

These conjectures was proved for $n = 2$ and no counter example was never found for greater values.

Conjecture (Maximal Utilization Bound)

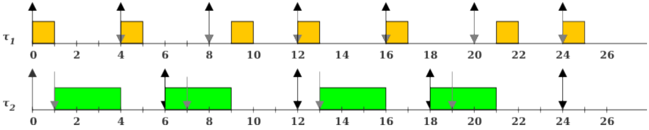
For any task set with total utilization less than or equal to 100% there exists a dual priority assignment that will meet all deadlines.

Conjecture (RM^2 Optimality)

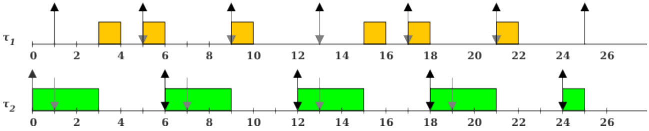
An optimal priority assignment could be RM^2:

- both P_i^1 and P_i^2 follow the RM rule, $\forall_{i,j}, P_i^1 < P_j^1$ iff $T_i < T_j$ and $P_i^2 < P_j^2$ iff $T_i < T_j$
- $\forall_{i,j}, P_i^2 < P_j^1$

Facts



(a) Synchronous Scenario



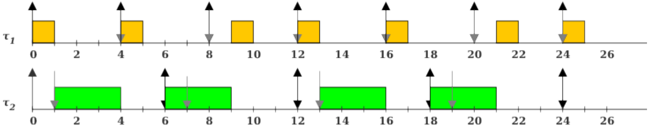
(b) Offset for task τ_1

Figure: Counter-intuitive properties illustrations

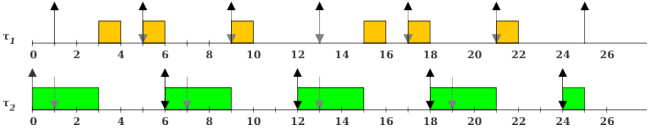
Property (Response time of the first job)

Consider a *synchronous* implicit-deadline task set, using dual priority the response time of the first job is not necessarily the largest one.

Facts



(a) Synchronous Scenario



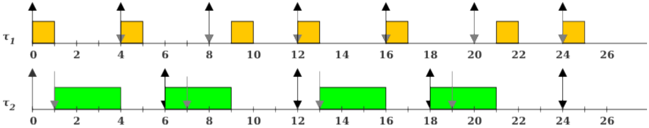
(b) Offset for task τ_1

Figure: Counter-intuitive properties illustrations

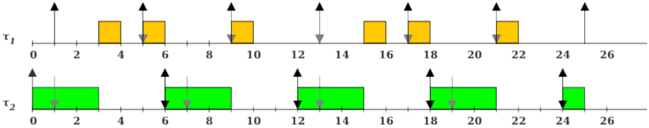
Property (The first busy period)

Consider a *synchronous* implicit-deadline task set, the first busy period is *not* a feasibility interval using dual priority scheduling.

Facts



(a) Synchronous Scenario



(b) Offset for task τ_1

Figure: Counter-intuitive properties illustrations

Property (No critical instant)

Considering dual priority scheduling, the synchronous case is not the worst case.

What about the implementation ?

- FP: Constant Time
- EDF: Can be implemented really efficiently but not in constant time
- DP: same as EDF

Interest ?

- Theoretical: scheduling algorithm classification.
- Practical : none ? Indeed the complexity is no less than the one of EDF.

What about the implementation ?

- FP: Constant Time
- EDF: Can be implemented really efficiently but not in constant time
- DP: same as EDF

Interest ?

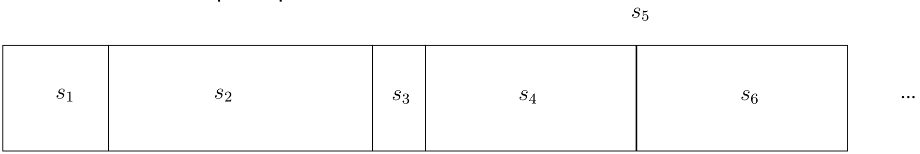
- Theoretical: scheduling algorithm classification.
- Practical : none ? Indeed the complexity is no less than the one of EDF.

But if the promotion points were not relative to releases, but dependent on the remaining costs ? And why only two priorities ?
Then, the implementation is constant time, like FP!

Fixed-K Priority Scheduler

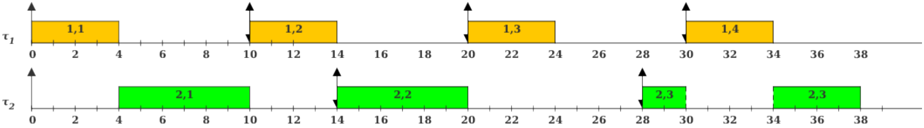
Early stages

From the beginning the idea was to increase the priority of the task when the remaining costs decreases.
Each task τ_i is cut in i segments of increasing priorities. The first segment is assigned the base priority (example DM priority), the second one the priority just above, and so on. Then the k^{th} segment correspond to code that the $k-1$ tasks with priorities just above the one of τ_i cannot preempt.

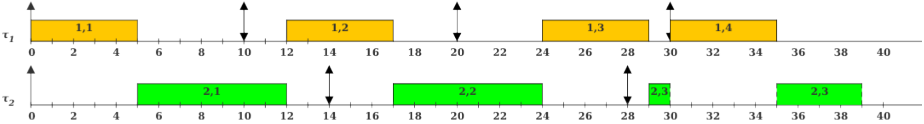


Motivating Example

$\tau_1 = (4, 10)$ and $\tau_2 = (6, 14)$ scheduled by DM



$\tau_1 = (5, 10)$ and $\tau_2 = (7, 14)$ scheduled by FKP



Intuitions

- Is the utilization bound 100% ? Harbour proved that cutting tasks into segments of different priorities increases the schedulability, and proved that the bound is 100% for two tasks.
- Issue: how to optimaly cut the task (size and priority of the segments) ?

Circular dependency!

to cut a task, we need to know the way other tasks are cut

- We need to be able to compute the worst case response time of task without knowledge on the way lowest base priority tasks are cut

Key idea 1

the first segment has a non null size

Then, a task cannot be preempted by a task with a lower "base" priority. Only be delayed.

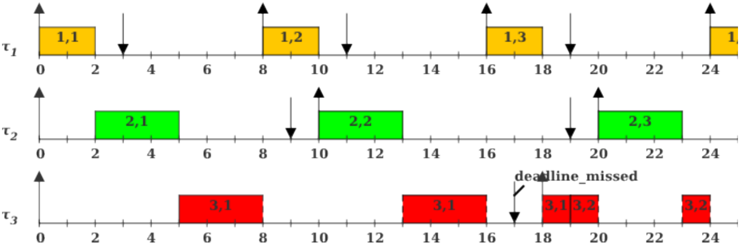
Intuitions

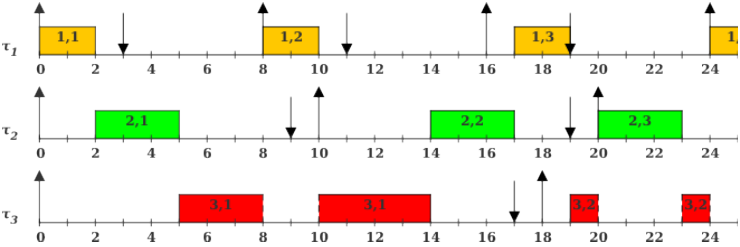
- We cut from the highest base priority one to the lowest base priority one
- Interference from lower priority tasks is just a blocking factor, that can happen only once in a level-i busy period
- This interference can be computed,

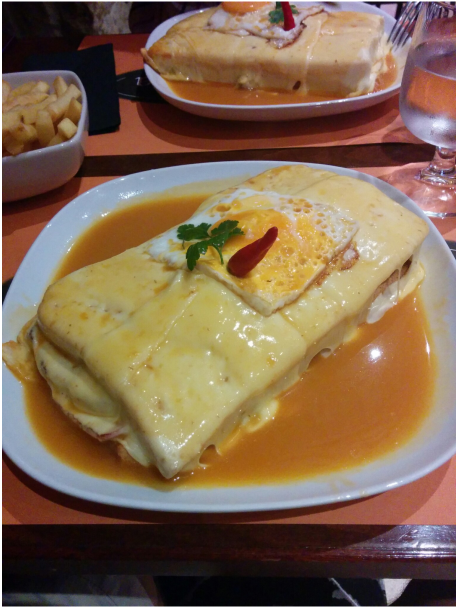
Key idea 2

because it cannot be worst than with DM

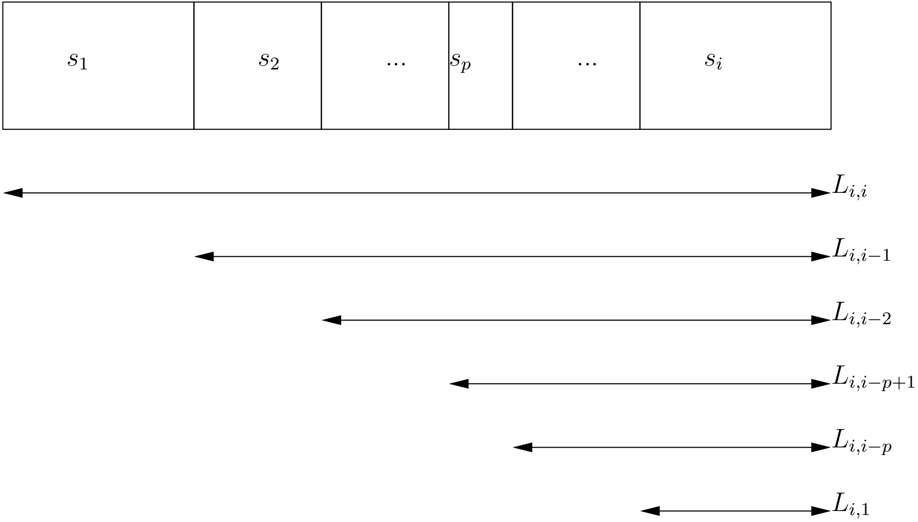
Illustration







Formally



Formally

$$L_{i,p} \stackrel{\text{def}}{=} \sum_{k=i-p+1}^{i} C_{i,k} \tag{1}$$

$$C_{i,k} \stackrel{\text{def}}{=} \begin{cases} L_{i,i-k+1} - L_{i,i-k} & k \neq i \\ L_{i,1} & k = i \end{cases} \tag{2}$$

Blocking Time

We denote by B_p the maximum blocking time that task τ_p can suffer due to the priority promotion of tasks with lower base priorities than τ_p.
Thanks to the non-null execution time of the first segment of every task τ_i, the following Lemma holds:

Lemma

Let τ_p be any task in τ. At most one job J_i^j such that $\mathrm{prio}(\tau_i) < \mathrm{prio}(\tau_p)$ can block the execution of a job of τ_p.

$$B_p \stackrel{\text{def}}{=} \max_{\tau_i \in \tau \setminus \mathrm{hp}(\tau_p)} \{L_{i,p}\} \tag{3}$$

Level-p Static Slack

Definition (Level-p Static Slack)

The Level-p static slack, denoted S_p, is the largest blocking time B_p that can be suffered by task τ_p such that τ_p remains schedulable with FKP.

It can be obtain by a dichotomy search, or adapting the classical slack time computation theory.

$$L_{i,p} = \begin{cases} C_i & p = i \\ \min(C_i - \epsilon, S_p) & p = i - 1 \\ \min(L_{i,p+1}, S_p) & \text{otherwise} \end{cases} \tag{4}$$

Response Time Analysis

There are several differences between FP and FKP that must be considered when computing the worst-case response time of a task τ_i

1. First, the jobs released by any task τ_i are composed of multiple segments. The response time of the j^{th} job J_i^j released by τ_i is therefore given by the sum of the response time of the segments of J_i^j.
2. Second, the segments composing τ_i have different priorities. It implies that different sets of tasks interfere with each segment.
3. Finally, as a consequence of the jobs increasing their priorities over time, tasks with lower base priorities than τ_i can block the execution of τ_i for B_i time units.

Response Time analysis

Lemma

The length $\mid \overline{BP}^{(i)} \mid$ *of the longest level-i busy period* $\overline{BP}^{(i)}$ *is the minimum solution to*

$$\mid \overline{BP}^{(i)} \mid = B_i + \sum_{r=1}^{i} \left\lceil \frac{\mid \overline{BP}^{(i)} \mid}{T_r} \right\rceil \times C_r \quad (5)$$

Lemma

The worst-case completion time $F_{i,k}^{j}$ *of the* k^{th} *segment of the* j^{th} *job released by* τ_i *in the longest level-i busy period is given by the minimum solution to*

$$\begin{aligned} F_{i,k}^{j} = & B_i + (j-1) \times C_i + \sum_{p=1}^{k} C_{i,p} \\ & + \sum_{q=i-k+1}^{i-1} \left\lceil \frac{F_{i,q}^{j}}{T_q} \right\rceil \times C_q + \sum_{r=1}^{i-k} \left\lceil \frac{F_{i,k}^{j}}{T_r} \right\rceil \times C_r \end{aligned} \quad (6)$$

Response Time analysis

$$F_{i,1}^{1} = B_i + C_{i,1} + \sum_{r=1}^{i-1} \left\lceil \frac{F_{i,1}^{1}}{T_r} \right\rceil \times C_r \quad (7)$$

$$F_{i,k}^{j} = F_{i,k-1}^{j} + C_{i,k} + I_{i,k}^{j} \quad (8)$$

$$I_{i,k}^{j} \stackrel{\text{def}}{=} \sum_{r=1}^{i-k} \left\lceil \frac{F_{i,k}^{j}}{T_r} \right\rceil \times C_r - \sum_{r=1}^{i-k} \left\lceil \frac{F_{i,k-1}^{j}}{T_r} \right\rceil \times C_r \quad (9)$$

Response Time analysis

Theorem

The worst-case response time of τ_i is given by

$$R_i = \max_{1 \le j \le n_i^{\max}} \left\{ F_{i,i}^{j} - (j-1) \times T_i \right\} \tag{10}$$

where $n_i^{\max} \stackrel{\text{def}}{=} \left\lceil \frac{|\overline{BP}^{(i)}|}{T_i} \right\rceil$ is the maximum number of jobs released by τ_i in $\overline{BP}^{(i)}$.

Fixed-K Priority Scheduler

Other Limited Preemption Techniques

Essentially, FKP algorithm can be classified amongst limited preemption techniques. Indeed, rising the priority between two segments leads to prevent some higher priority tasks to preempt the task.
There is three main approaches:

- **Preemption Thresholds Scheduling (PTS)**a parameter called preemption threshold is added to each task. Only tasks with a priority higher than the running task threshold can preempt it.
- **Deferred Preemptions Scheduling (DPS)**a longest non preemptive interval is defined for each task. When an higher priority task arrives, the preemption can be delayed for that interval.
- **Fixed Preemption Points (FPP)**specific preemption point are defined by the programmer in the task. Then at runtime, preemption are delayed until the next preemption point.

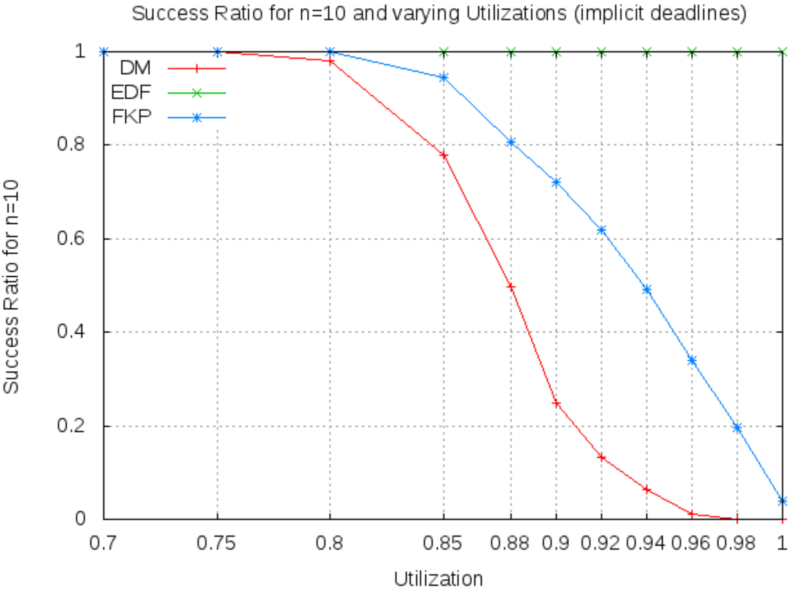
Fixed-K Priority Scheduler

Evaluation

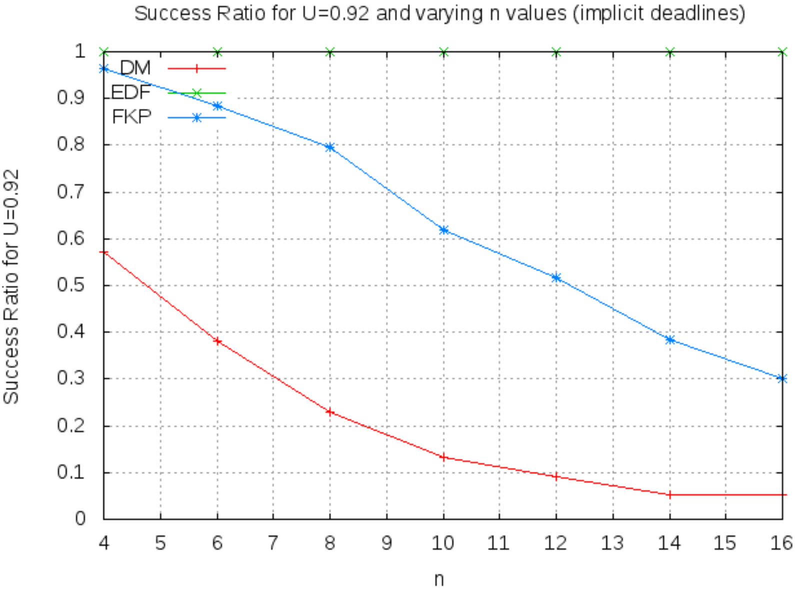
We evaluate the performances of FKP against DM and EDF regarding the following metrics:

- *Success ratio*, the number of schedulable systems divided by the number of systems,
- *WCRT ratio*, for each task, the worst case response time is normalised with respect to the task deadline, then an average value for the system is computed, and we present the average value over the simulated systems,
- *Max Tardiness ratio*, for each task, the maximum tardiness is normalised with respect to the task deadline, then an average value is computed for the system, and we present the average value over the simulated systems,
- *Preemptions ratio*, the total number of preemptions is divided by the total number of jobs released during the simulation, and we present the average value over the simulated systems.

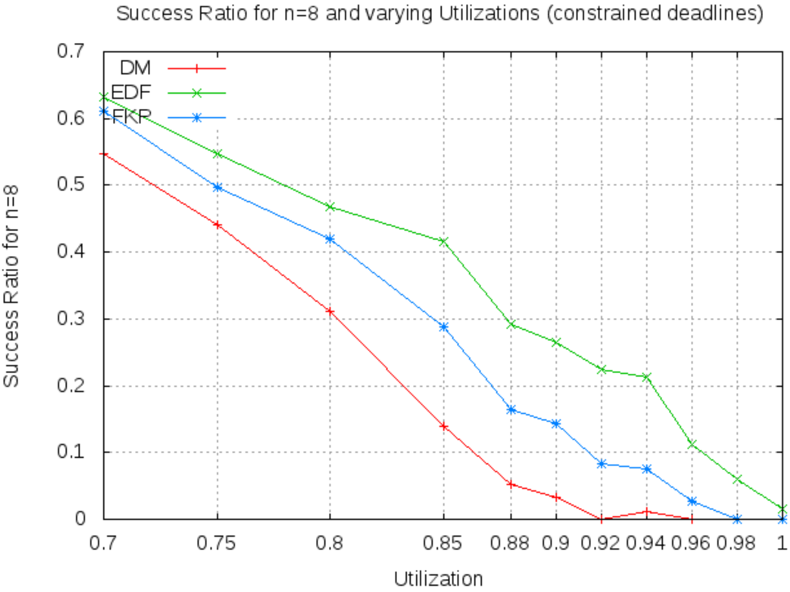
Evaluation


Success Ratio for n=10 and varying Utilizations (implicit deadlines)
DM
EDF
FKP
Success Ratio for n=10
0
0.2
0.4
0.6
0.8
1
0.7
0.75
0.8
0.85
0.88
0.9
0.92
0.94
0.96
0.98
1
Utilization

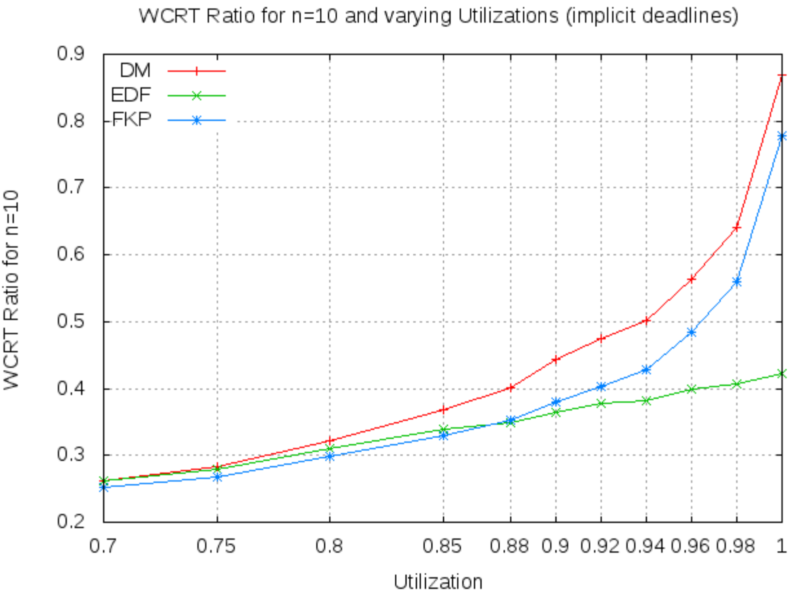
Evaluation


Success Ratio for U=0.92 and varying n values (implicit deadlines)
Success Ratio for U=0.92
n
DM
EDF
FKP
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
4
5
6
7
8
9
10
11
12
13
14
15
16

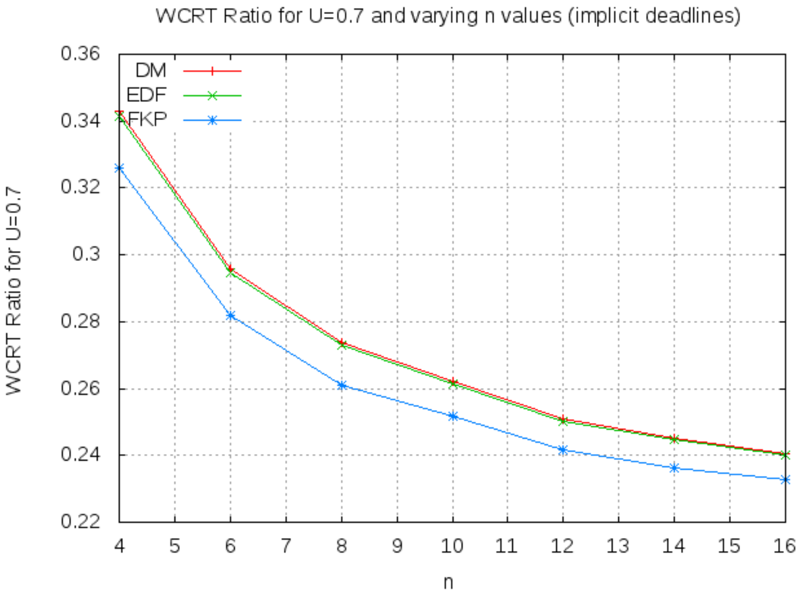
Evaluation


Success Ratio for n=8 and varying Utilizations (constrained deadlines)
DM
EDF
FKP
Success Ratio for n=8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.7 0.75 0.8 0.85 0.88 0.9 0.92 0.94 0.96 0.98 1
Utilization

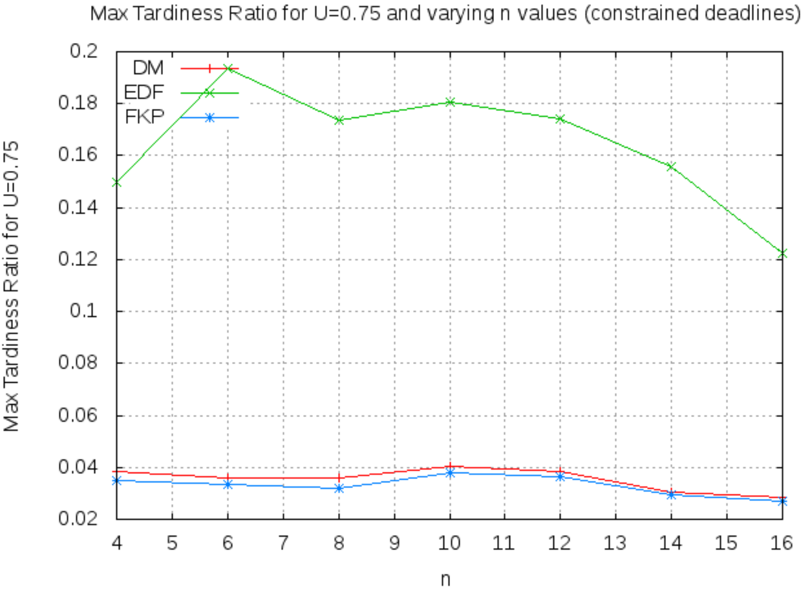
Evaluation


WCRT Ratio for n=10 and varying Utilizations (implicit deadlines)
DM
EDF
FKP
WCRT Ratio for n=10
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.7
0.75
0.8
0.85
0.88
0.9
0.92
0.94
0.96
0.98
1
Utilization

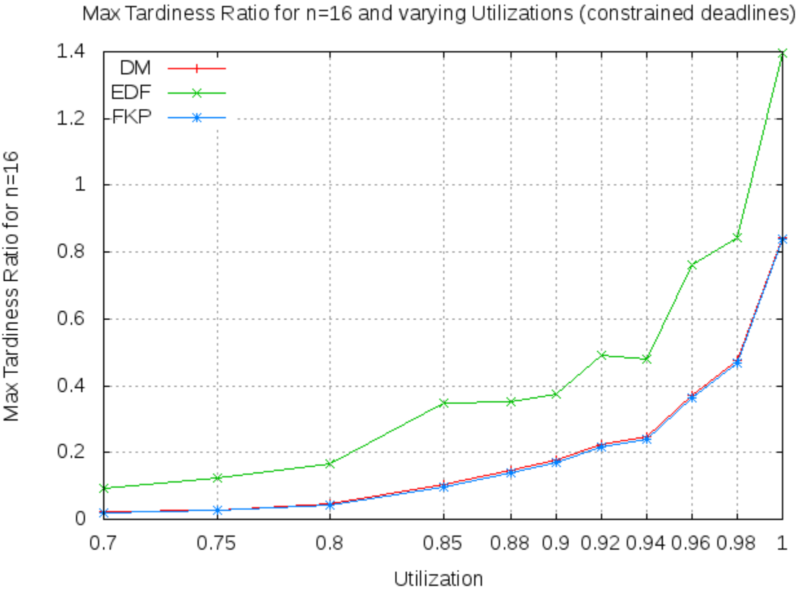
Evaluation


WCRT Ratio for U=0.7 and varying n values (implicit deadlines)
DM
EDF
FKP
WCRT Ratio for U=0.7
0.36
0.34
0.32
0.3
0.28
0.26
0.24
0.22
4
5
6
7
8
9
10
11
12
13
14
15
16
n

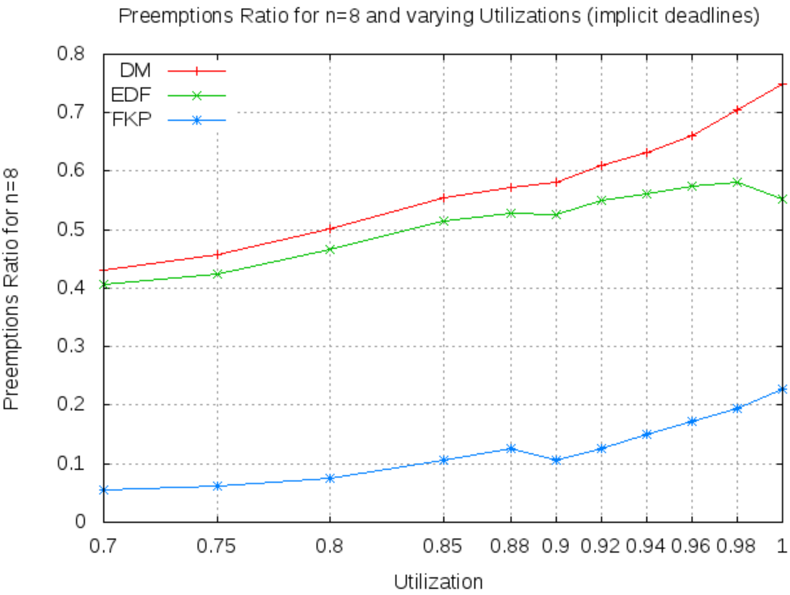
Evaluation


Max Tardiness Ratio for U=0.75 and varying n values (constrained deadlines)
DM
EDF
FKP
Max Tardiness Ratio for U=0.75
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
4
5
6
7
8
9
10
11
12
13
14
15
16
n

Evaluation

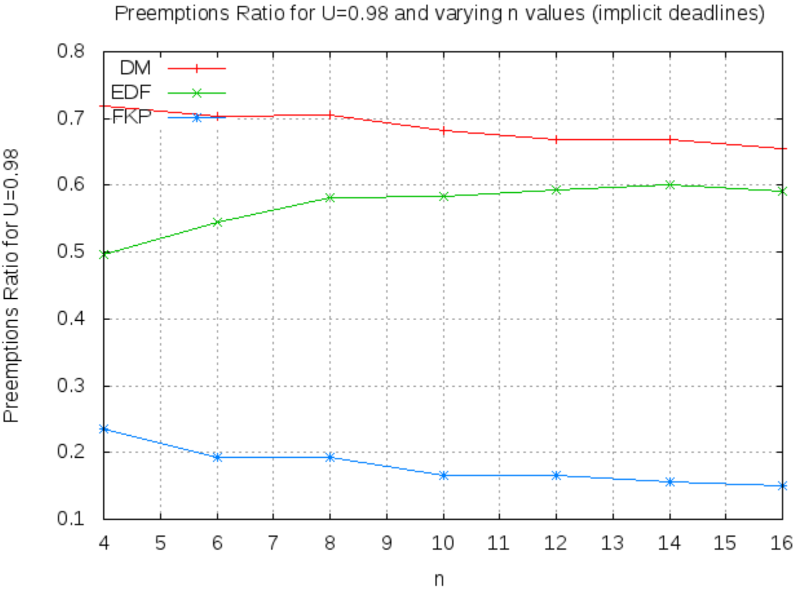

Max Tardiness Ratio for n=16 and varying Utilizations (constrained deadlines)
DM
EDF
FKP
Max Tardiness Ratio for n=16
Utilization
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0.7
0.75
0.8
0.85
0.88
0.9
0.92
0.94
0.96
0.98
1

Evaluation



Preemptions Ratio for n=8 and varying Utilizations (implicit deadlines)
DM
EDF
FKP
Preemptions Ratio for n=8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.7
0.75
0.8
0.85
0.88
0.9
0.92
0.94
0.96
0.98
1
Utilization

Evaluation


Preemptions Ratio for U=0.98 and varying n values (implicit deadlines)
DM
EDF
FKP
Preemptions Ratio for U=0.98
n
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
4
5
6
7
8
9
10
11
12
13
14
15
16

Fixed-K Priority Scheduler

Future Work / conclusion

- FKP is a constant time scheduler
- higher schedulability compare to FP
- less preemptions compare to FP and EDF
- easy to implement, but need offline computations (could be a problem for online admission in dynamic systems)

As future works, we would like to

- evaluate performances against other limited preemption techniques
- formalise a utilisation bound for FKP
- provide an actual implementation of this algorithm in an embedded real-time operating system
- extend the task model to consider tasks that share resources and generalise it to multiprocessor platforms.