Ordonnancement sous Posix1003.1b

Nicolas Navet

INRIA - Projet TRIO
http://www.loria.fr/~nnavet

Posix1003.1b : standard pour les Unix temps réel

- **POSIX :** famille de normes IEEE (comité 1003) visant à standardiser les services et interfaces offerts par les OS de type Unix. Ex: Posix 1003.1-librairie C, Posix 1003.2-Shell, Posix 1003.1b-temps réel, Posix 1003.1c-threads

- **Objectif :** portabilité au niveau du code source des applications TR

- **Historique :** début des travaux en 1985, Posix 1003.1 en 1990, 1003.1b en 1993 (aka “posix.4”), 1003.1c en 1995, seconde version de Posix 1003.1b en 1996.

- **Posix 1003.1b définit des fonctionnalités “temps réel” :** sémaphores, verrouillage en RAM de processus, fichiers “mappés” en mémoire, E/S asynchrones, timers et horloges, signaux, **ordonnancement**

Posix 1003.1b : politiques d'ordonnancement (1/2)

✔ Posix 1003.1b spécifie 3 politiques: **SCHED_FIFO, SCHED_OTHER, SCHED_RR**

SCHED_FIFO :

- Ordonnancement préemptif à priorités (32 niveaux de priorités minimum)
- FIFO parmi les instances de même priorité
- Sous réserve
  - que la priorité de chacune des instances d'une même tâche récurrente soit identique et fixe au cours du temps
  - qu'une priorité ne soit allouée qu'à une seule tâche récurrente

  ⇔ **Fixed Priority Preemptive (FPP)**

SCHED_OTHER :

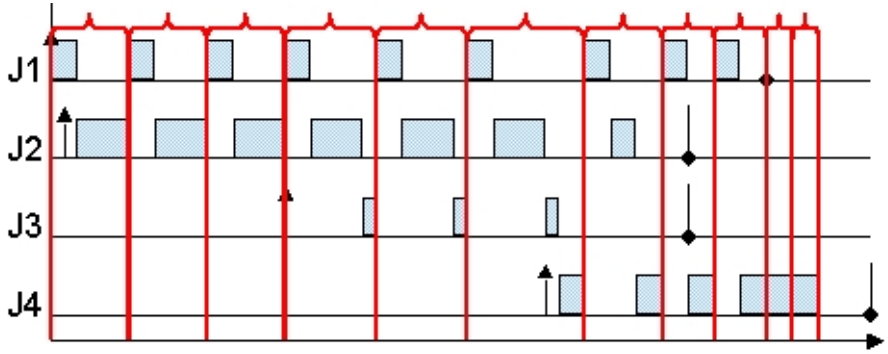
- non-définie par le standard mais disponibilité requise
- pointe généralement sur SCHED_FIFO où une politique de type "time sharing"

Posix 1003.1b : politiques d'ordonnancement (2/2)

SCHED_RR : politique de type Round-Robin

- les processus de même priorité se partagent le CPU
- un processus utilise un quantum de temps puis est déplacé en queue de la file correspondant à sa priorité
- il y a préemption par les tâches plus prioritaires
- la valeur du quantum de temps peut être globale sur le système (fixe ou configurable) où spécifique pour une tâche où un niveau de priorités

✔ Un cycle Round-Robin : intervalle t.q. chaque tâche active τ_k utilise une fois son quantum Ψ_k (et $\overline{\Psi}_k = \sum \Psi_i - \Psi_k$)



Posix 1003.1b : organisation en couches de tâches

✔ On peut identifier des **couches avec des tâches de même politique** - les couches sont priorisées entre elles (ici $\lambda_1 \succ \lambda_2 \succ \lambda_3$)

Couche	m_l	Tâches	Politique	Priorité	Quantum
λ_1 (FPP)		τ_1	SCHED_FIFO	**1**	-
		τ_2	SCHED_FIFO	**2**	-
		τ_3	SCHED_FIFO	**3**	-
	$m_1 = 4$	τ_4	SCHED_FIFO	**4**	-
λ_2 (RR)		τ_5	SCHED_RR	**5**	15
		τ_6	SCHED_RR	**5**	5
	$m_2 = 7$	τ_7	SCHED_RR	**5**	10
λ_3 (RR)		τ_8	SCHED_FIFO	**6**	-
		τ_9	SCHED_FIFO	**7**	-
		τ_{10}	SCHED_FIFO	**8**	-
		τ_{11}	SCHED_FIFO	**9**	-
	$m_3 = 12$	τ_{12}	SCHED_FIFO	**10**	-

➠ Les instants d'exécution d'une tâche ne dépendent que de la composition des couches plus prioritaires et non des choix d'ordonnancement dans ces couches

Posix 1003.1b : l'API pour l'ordonnancement

```
#define _POSIX_PRIORITY_SCHEDULING /* si support Posix 1003.1b */
```

✔ Paramètres de la politique d'ordonnancement pour une tâche:

```
struct sched_param {
int sched_priority;
... };
```

✔ Prototypes des fonctions:

```
int sched_setparam (pid_t pid, struct sched_param *param);
int sched_getparam (pid_t pid, struct sched_param *param);
int sched_setscheduler (pid_t pid, int policy, struct sched_param *param);
int sched_getscheduler (pid_t pid);
int sched_yield (void);
int sched_get_priority_max (int policy);
int sched_get_priority_min (int policy);
int sched_rr_get_interval (pid_t pid, struct timespec *t);
```

Pour conclure sur la spécification de l'ordonnancement sous Posix 1003.1b

---

+ Facilite réellement la portabilité au niveau source car simple et largement adoptée

+ Offre un bon support pour l'ordonnancement à priorités fixes

- Ne propose aucun support pour EDF, pas de services d'activation périodique des tâches, Priority Ceiling Protocol non standardisé

- Les plages de priorités pour les $\neq$ politiques ne sont pas spécifiées : souvent disjointes avec priorités SCHED_RR $\leq$ priorités SCHED_FIFO

- La taille du quantum et la possibilité de le changer dépend de l'OS

➠ L'idée générale est que RR n'a d'utilité qu'aux niveaux de priorités les + faibles ? Qu'en est-il ?

Analyse d’ordonnançabilité d’un ensemble de tâches récurrentes sous Posix 1003.1b

---

1. **Temps de réponse d’une tâche dans une couche FPP**

2. **Temps de réponse d’une tâche dans une couche RR**

Bornes sur le temps de réponse dans une couche FPP (1/2)

Notation : $W_{I_k}(t)$ est la quantité de travail en attente d'exécution à l'instant t dont la priorité est $\geq$ à la priorité de τ_k

Définition 1: **une période d'interférence de niveau** k est un intervalle de temps $[U_k, E_k[$ t.q. qu'à chaque instant dans $]U_k, E_k[$, $W_{I_k}(t) > 0$ avec $W_{I_k}(U_k) = 0$ et $W_{I_k}(E_k) = 0$

Théorème (Lehoczky): sous FPP, le temps de réponse de toute instance d'une tâche τ_k est borné par le temps de réponse d'une instance de τ_k dans la première période d'interférence de niveau k du scénario d'activation majorant (i.e. après un instant critique)

$$R_{k,n} \leqslant \hat{R}_k = \max_{j=1,2,\ldots \text{ t.q. } a_{k,j} < e_{k,j-1}} r_{k,j}.$$

Etapes de la preuve : 1) $R_{k,n}$ est borné par le temps de réponse d'une instance de τ_k dans le scénario d'activation majorant notée $\tau_{k,\tilde{n}}$ 2) $\tau_{k,\tilde{n}}$ se situe nécessairement dans la première période d'interférence de niveau k du scénario majorant

Bornes sur le temps de réponse dans une couche FPP (2/2)

Notations : $\hat{S}_{1..k-1}(x)$ est une fonction majorante du travail des tâches plus prioritaires que τ_k, $\hat{S}_{k,1..n}(x)$ majore le travail des instances $\tau_{k,1}$, $\tau_{k,2}$,...,$\tau_{k,n}$

✔ Calcul de $e_{k,n}$ avec $\tau_{k,n}$ dans la première période d'interférence :

$$e_{k,n} \quad = \quad \min\{x > 0 \mid \hat{S}_{1..k-1}(x) + \hat{S}_{k,1..n}(x) = x\}$$

✔ Par exemple, pour des tâches périodiques :

$$e_k^0(n) \quad = \quad 0, \quad e_k^q(n) = n \cdot c_k + \sum_{\tau_j \succ \tau_k} \left\lceil \frac{e_k^{q-1}(n)}{T_j} \right\rceil \cdot c_j$$

✔ $\hat{R}_k = \max_{n=1,2,..}(e_{k,n} - a_{k,n})$ avec $a_{k,n} = (n-1) \cdot T_k$

✔ En partant de $n = 1$, on calcule $e_{k,n}$ jusqu'au moment où $e_{k,n} \leq a_{k,n+1} = n \cdot T_k$

Periode d'interférence sous RR

Définition 2 [3]: **une RR-période d'interférence de niveau k** est un intervalle de temps $[U_k^{RR}, E_k^{RR}[$ t.q.

- à chaque instant dans $]U_k^{RR}, E_k^{RR}[$, il y a du travail d'une couche plus prioritaire ou du travail d'une instance de τ_k en attente d'exécution

- en U_k^{RR} et E_k^{RR}, il n'y a pas de travail plus prioritaire ou de travail d'une instance de τ_k en attente

- une instance de τ_k au moins est mise à disposition dans $[U_k^{RR}, E_k^{RR}[$

nb: une RR-période d'interférence se termine avec la fin d'exécution d'une instance de τ_k mais peut commencer soit en l'arrivée d'une instance de τ_k ou d'une instance dans une couche + prioritaire

Temps de réponse sous RR : le principe

✔ Soit une instance $\tau_{k,n}$ dans la couche RR m_l mise à disposition dans la RR-période d'interférence $[U_k^{RR}, E_k^{RR}[$. Avant son instant de fin d'exécution, il faut exécuter :

1. son travail + le travail des instances de τ_k arrivées avant :
   $S_k^n = S_k(U_k^{RR}, A_{k,n}) + C_{k,n}$ soit au plus $\lceil \frac{S_k^n}{\Psi_k} \rceil$ cycles RR

2. le travail des couches plus prioritaires : $S_{1..m_{l-1}}(U_k^{RR}, t)$

3. le travail des autres instances de la couche RR : le minimum entre
   (a) la somme des quanta des autres tâches de la couche : $\lceil \frac{S_k^n}{\Psi_k} \rceil \cdot \overline{\Psi}_k$
   (b) le travail total à exécuter pour les autres tâches de la couche : $\overline{W}_k(U_k^{RR}) + \overline{S}_k(U_k^{RR}, t)$

$$E_{k,n} \leqslant E^*_{k,n} = \min\{t > U_k^{RR} \mid \min\left(\left\lceil \frac{S_k^n}{\Psi_k} \right\rceil \cdot \overline{\Psi}_k, \overline{W}_k(U_k^{RR}) + \overline{S}_k(U_k^{RR}, t)\right) + S_{1..m_{l-1}}(U_k^{RR}, t) + S_k^n = t - U_k^{RR}\}$$

Borne sur le temps de réponse sous RR : le principe

Théorème [3]: sous RR, le temps de réponse de toute instance d'une tâche τ_k est inférieur au maximum des bornes sur les temps de réponse des instances de τ_k dans la première RR-période d'interférence de niveau k du scénario d'activation majorant (i.e. après un instant critique)

➠ En pratique, sur-estimation de l'ordre de 15% par rapport au maximum sur une simulation

Temps de réponse d'un ensemble de jobs sous RR [5]: algorithme exact en $O(n^2)$ basé sur l'identification des $\neq$ cycles RR

➠ peut être utilisé pour des tâches périodiques à décalages initiaux fixés

➠ valable pour des tâches périodiques à décalages initiaux non-fixés ??

Sections critiques et Priority Ceiling Protocol sous Posix 1003.1b

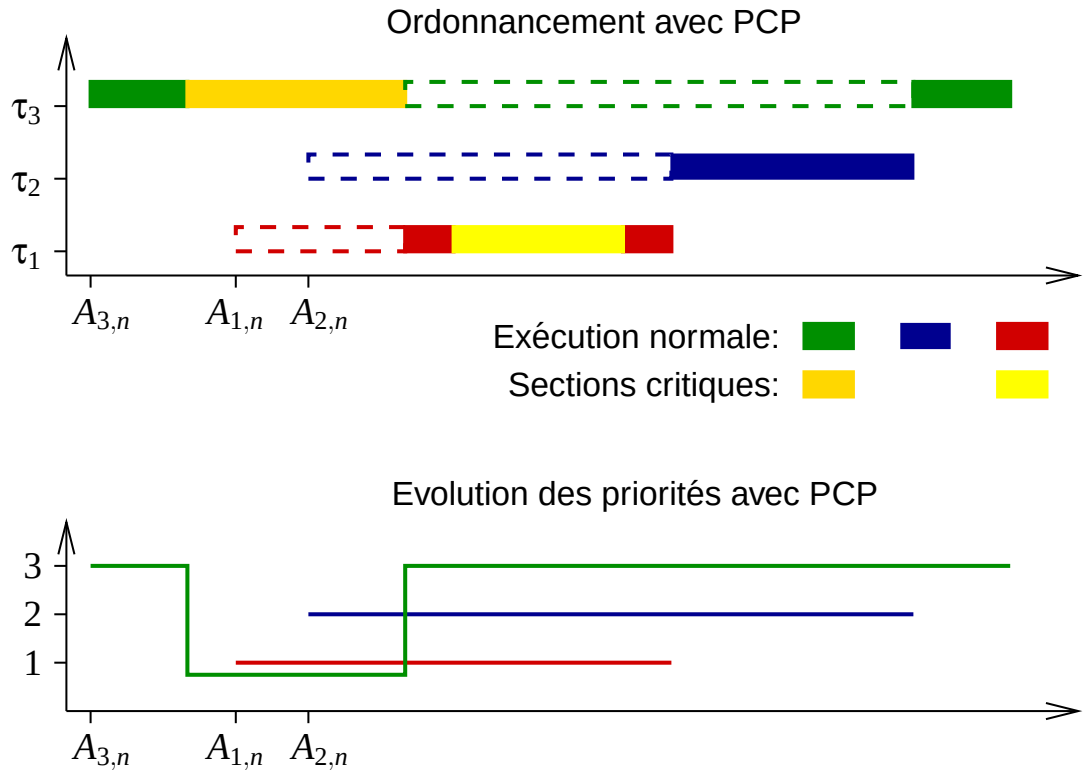
1. **Rappel : le PCP sous FPP**

2. **PCP sous RR : problème et solutions**

Priority Ceiling Protocol (PCP) sous FPP

✔ PCP : quand une tâche entre en section critique, elle prend la priorité de la tâche de plus forte priorité susceptible d'utiliser la ressource

✔ 3 tâches τ_1, τ_2 et τ_3, ressource partagée entre τ_1 et τ_3 avec $\tau_1 \succ \tau_2 \succ \tau_3$



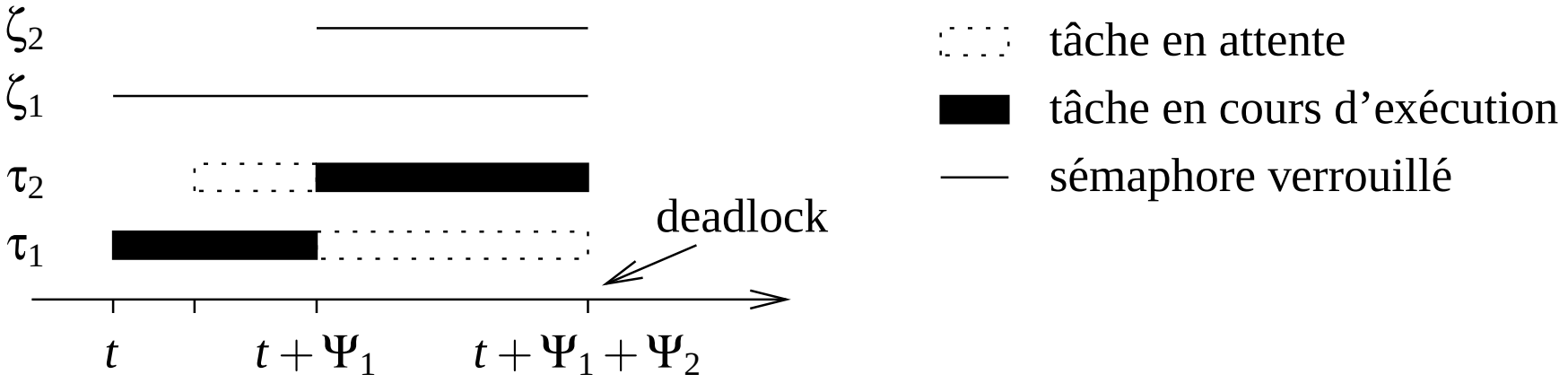
PCP sous RR : le problème

Observation 1 : PCP sans effet pour des tâches de même priorité sous RR

Observation 2 : les sémaphores seules conduisent à des deadlocks sous RR

τ_1 a besoin de ζ_1 puis de ζ_2

τ_2 a besoin de ζ_2 puis de ζ_1



PCP sous RR : une première solution

Proposition 1 (PCP-RR.1) [3]: le quantum de temps d'une tâche est ignoré lorsque la tâche est en section critique

+ simplicité d'implémentation

➠ changer la taille du quantum si possible ou

➠ augmenter la priorité en entrée de section critique au niveau immédiatement supérieur à la tâche de plus forte priorité susceptible d'utiliser la ressource (niveau non utilisé par ailleurs)

- distorsion vis-à-vis de l'équité de RR

- pires temps de réponse : dans chaque cycle RR, une tâche τ_k peut utiliser $\Psi_k + z_k$ unités de temps où Ψ_k est la taille du quantum et z_k la taille de la plus grande section critique de τ_k

➠ PCP-RR.1 est acceptable si la taille des sections critiques est petite par rapport à la taille du quantum

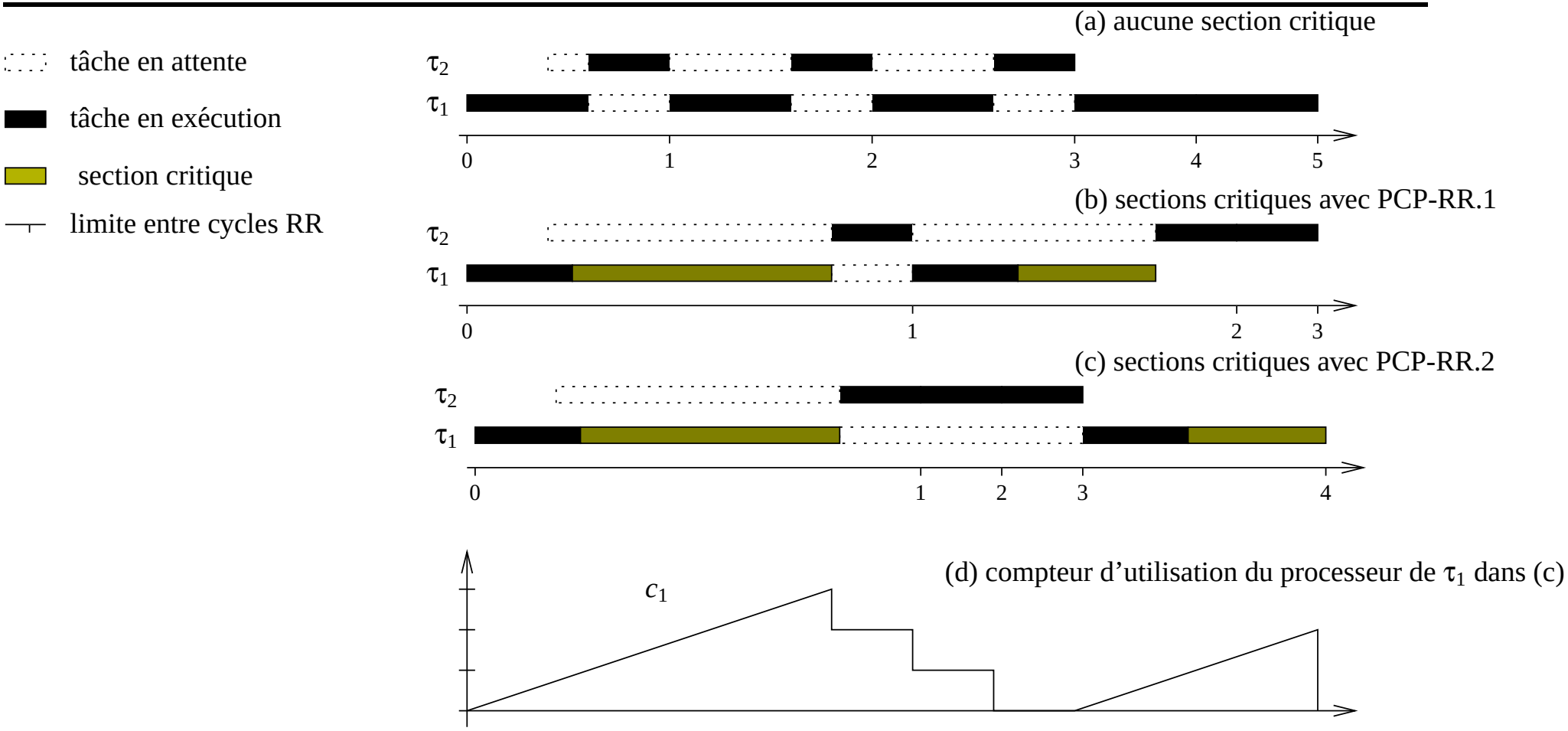
PCP sous RR : une meilleure solution

Proposition 2 (PCP-RR.2) [3]: PCP-RR.1 + “remboursement” du trop perçu ultérieurement. Un compteur d’utilisation c_k est incrémenté lorsque la tâche est exécutée.

1. **Sélection d’une tâche :** soit τ_k la prochaine tâche dans le cycle RR
   (a) si $c_k < \Psi_k$ alors la plus ancienne instance de τ_k est choisie
   (b) si $c_k \geq \Psi_k$ alors $c_k := c_k - \Psi_k$ et τ_k n’est pas choisie

2. **Arrêt de la tâche en cours d’exécution :** $\tau_{k,n}$ en cours d’exécution
   (a) $\tau_{k,n}$ en section critique lorsque c_k devient plus grand que Ψ_k alors $\tau_{k,n}$ interrompu à la sortie de la section critique et $c_k := c_k - \Psi_k$
   (b) $\tau_{k,n}$ pas en section critique lorsque c_k devient plus grand que Ψ_k alors $\tau_{k,n}$ interrompu et $c_k = 0$
   (c) $c_k \leq \Psi_k$ quand $\tau_{k,n}$ se termine et $\tau_{k,n+1}$ non présente alors $c_k = 0$

PCP sous RR : exemple d'ordonnancement



➟ Sous PCP-RR.2, une tâche τ_k ne peut jamais avoir utilisée plus de z_k unités de temps CPU en plus que prévu ..

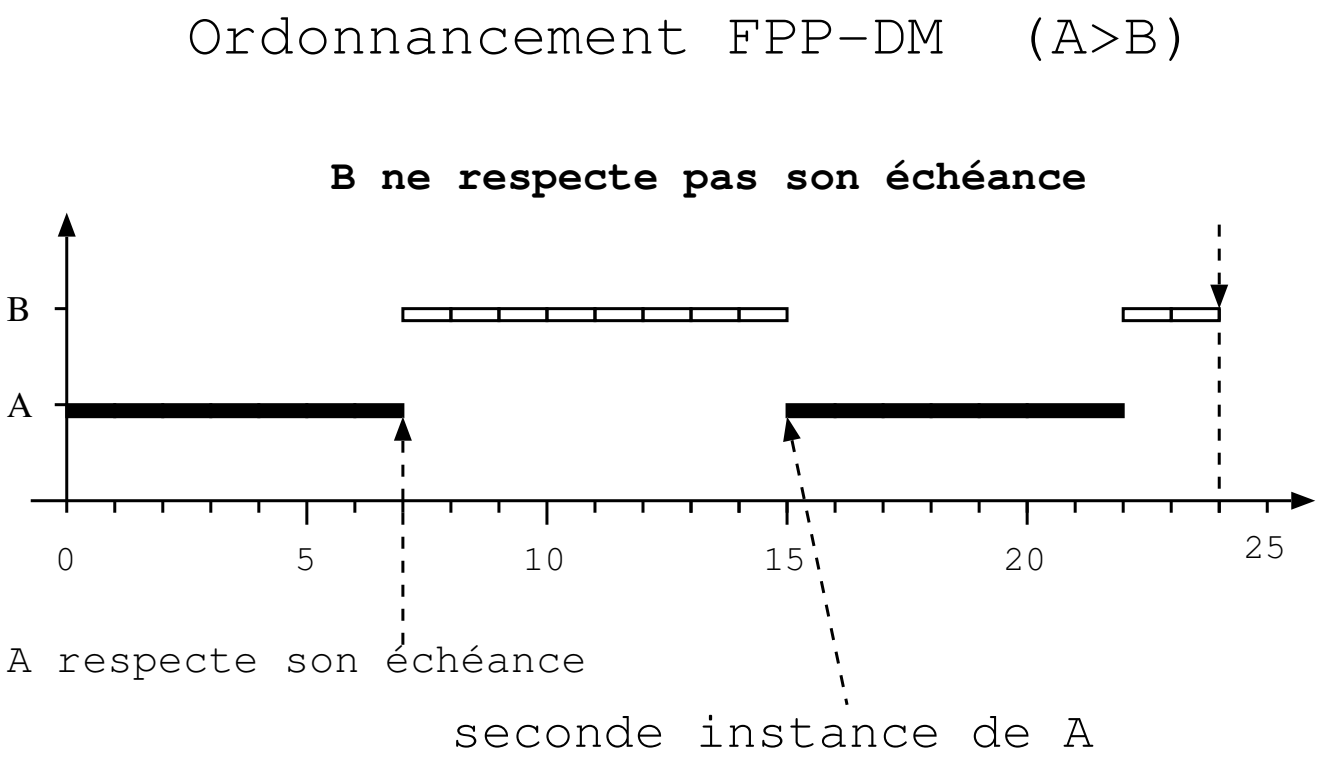
Utilité de RR pour le temps réel

1. **Augmentation de l'ordonnançabilité du système**

2. **Satisfaction de critères secondaires**

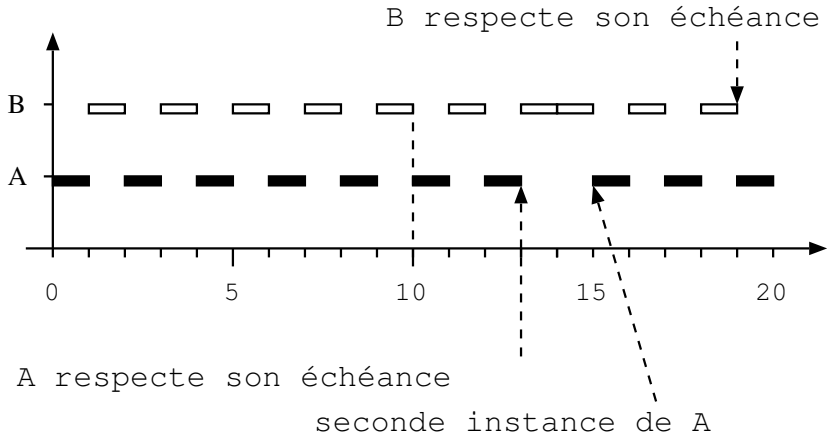
RR et ordonnançabilité du système (1/3)

- ✔ Tâche A $(C = 7, \overline{D} = 15, T = 15)$ - tâche B $(C = 10, \overline{D} = 20, T = 50)$
- ✔ Sous FPP, l'allocation Deadline Monotonic (DM) est optimale pour $\overline{D} \leq T$

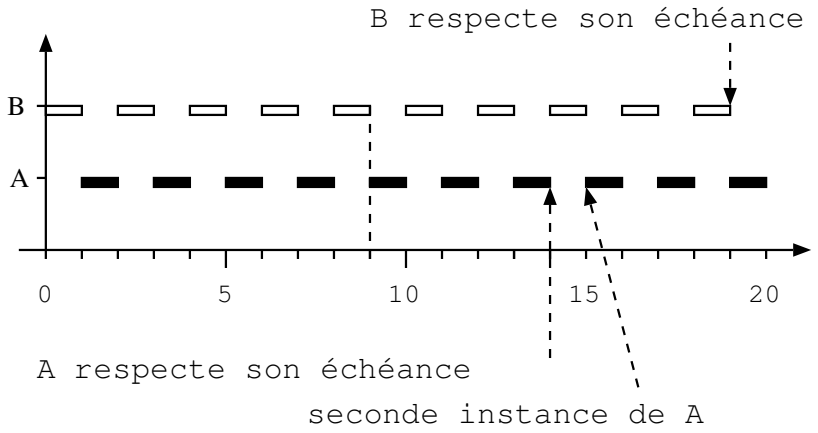


RR et ordonnançabilité du système (2/3)

Ordonnancement RR 1



Ordonnancement RR 2



RR et ordonnançabilité du système (3/3)

✔ Sous Posix1003.1b, choisir les priorités et les politiques est un problème combinatoire :

- partition d'un ensemble de n tâches en k niveaux de priorités (=sous-ensembles non-vides) : $\frac{1}{k!}\sum_{i=0}^{k}(-1)^{(k-i)}\binom{k}{i}i^n$

- la politique est induite par la cardinalité du sous-ensemble : $\# = 1$ implique FPP, $\# > 1$ implique RR

- il y a $k!$ possibilités de prioriser les k sous-ensembles avec $k = 1, 2, \ldots$ou n

soit au total $\sum_{k=1}^{n}\sum_{i=0}^{k}(-1)^{(k-i)}\binom{k}{i}i^n$ possibilités d'allouer priorités et tâches (10^8 pour 10 tâches)

Expérimentation 1 [4] : une recherche aléatoire avec 2000 essais permet de rendre faisable 15% des ensembles de tâches non-faisables sous FPP en utilisant FPP + RR

➠ On peut vraisemblablement trouver de bonnes solutions heuristiques ou un algorithme branch & bound optimal ..

Satisfaction de critères secondaires

✔ Typiquement, pour des applications de contrôle-commande, il est souhaitable de:

1. minimiser la gigue sur les fins d'exécution des tâches
2. maximiser la fraîcheur temporelle de données produites par des tâches en entrée d'un algorithme
3. maximiser la cohérence temporelle de données en entrée d'un algorithme

Expérimentations 2 [4]: recherche avec un algorithme génétique - mesure par simulation - amélioration obtenue par rapport à FPP en utilisant RR+FPP

résultats: critère 1: mieux dans 36% des cas - critère 2: mieux dans 68% des cas - critère 3: mieux dans 70% des cas

Expérimentations 3 [6]: multi-tâches avec 3 boucles de régulation - simulations sous mathlab/simulink - critères: temps de réponse du système et dépassement

résultats: RR mieux que FPP en moyenne mais inférieur à une politique Round-Robin sur les priorités

Ordonnancement sous contraintes de temps et d'énergie

Contexte:

GSM

Ordinateurs portables

Implants medicaux

satellites

1. **Typologie des solutions**

2. **L'existant sous Posix**

Typologie des solutions et existant

Ordonnancement faible consommation : fonctionner à la fréquence la + faible qui garantisse le respect des contraintes de performances

Techniques et solutions existantes : (Posix 1003.1b = utilisation conjointe de FPP + RR)

1. **Fréquence unique sur le système : FPP - RR - Posix 1003.1b**
2. **Fréquence unique par tâche : FPP**
3. **Fréquence unique par instance : FPP**
4. **Vitesse choisie par cycle d'horloge : FPP**
5. **Redistribution des temps d'exécution non-utilisés** (avec ou sans instrumentation du code) : **FPP**

Fréquence unique sous Posix 1003.1b

Résultat 1: la fréquence de fonctionnement sous Posix est nécessairement plus élevée ou égale à la fréquence minimale admissible sous EDF (S_{edf})

Algorithme (tâches récurrentes): en partant de S_{edf}, choisir la plus petite vitesse disponible sur le CPU t.q. la borne sur le temps de réponse de chaque tâche soit inférieure à l'échéance

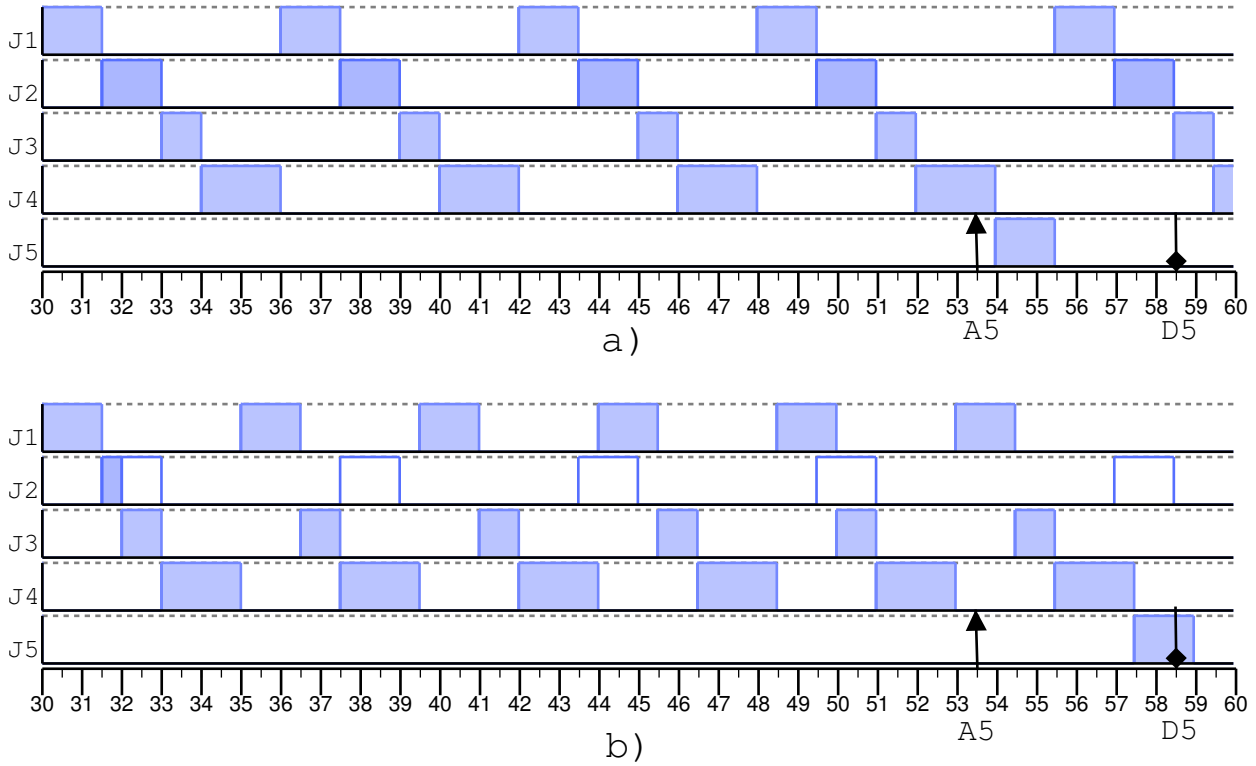
➠ **Pb :** réduction de consommation sous-optimale car calcul de bornes sur le temps de réponse pessimiste sous RR

✔ Il existe un algorithme [5] optimal dans le cadre de tâches récurrentes à décalages initiaux fixés ou de jobs ordonnancés dans une même couche RR, on observe que:

1. la terminaison d'un job plus tôt que son WCET entraîne parfois le non-respect des échéances
2. la faisabilité est non-monotone en la vitesse

Ordonnancement RR low-power (1/2)

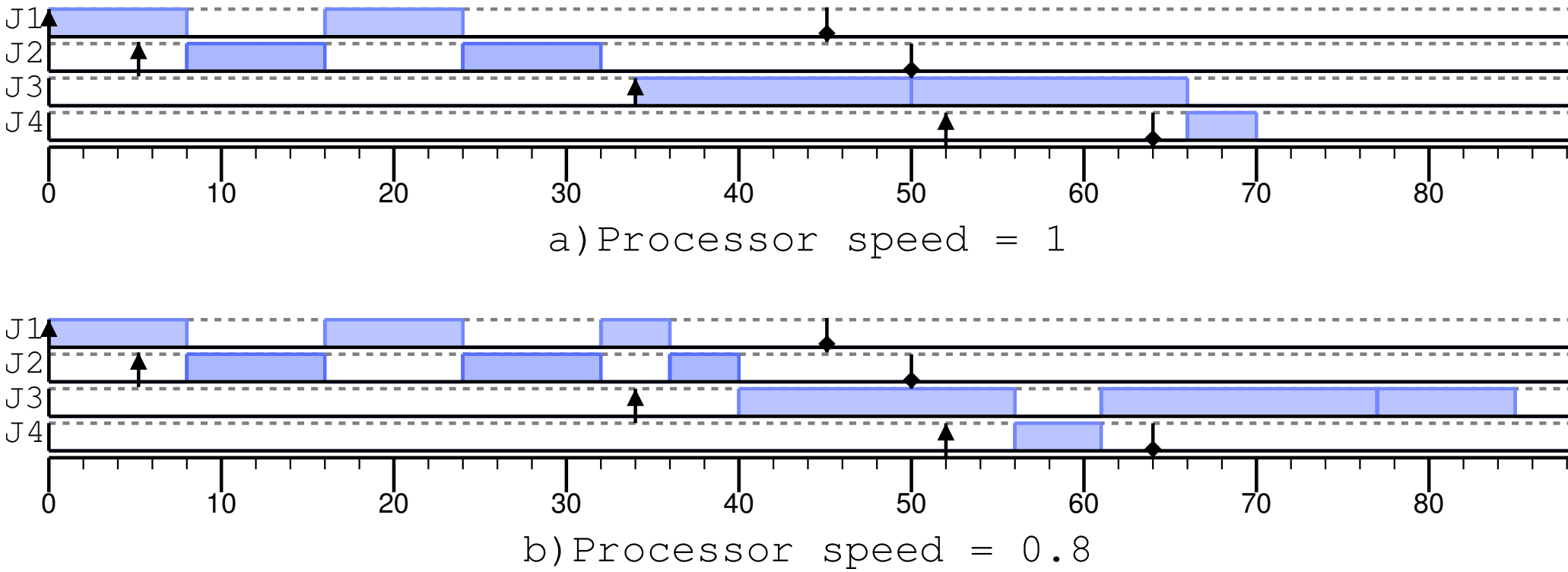
Observation 1: la terminaison d'un job plus tôt que son WCET peut conduire à la non-faisabilité du système



✔ Des mécanismes en-ligne existent [5] pour prévenir ce phénomène ..

Ordonnancement RR low-power (2/2)

Observation 2: la faisabilité n'est pas monotone en la fréquence



✔ Il faut tester la faisabilité en partant de la fréquence la plus faible jusqu'à S_{edf}

Conclusions / Perspectives

✔ Posix 1003.1b fournit un bon support pour l'ordonnancement à priorités fixes

✔ mais aucune garantie sur les performances temps réel de l'OS (ex: appels systèmes préemptibles)

✔ la politique RR est moins bien spécifiée et mal exploitée en pratique

Perspectives:

- ➟ borner la surestimation dans le calcul des bornes sur le temps de réponse sous RR
- ➟ solutions efficaces pour fixer tâches et priorités avec quantum de taille fixe et variable (se rapprocher d'EDF en faisabilité?)
- ➟ algorithmes et mécanismes exécutifs pour l'ordonnancement temps réel sous contrainte d'énergie

Logiciels

Sont téléchargeables à l'adresse http://www.loria.fr/~nnavet :

- ✔ RTS et TkRTS écrit par J. Migge pour calculer des bornes sur les temps de réponse pour FPP,NP-FPP,EDF,NP-EDF,RR
- ✔ un simulateur d'ordonnancement (applet Java - chronogramme) pour toute politique indépendante du temps (i.e. priorité d'une instance fixe)
- ✔ un ordonnanceur de niveau "applicatif" avec un service d'activation périodique configurable pour toute politique indépendante du temps

Références

[1] ISO/IEC, “9945-1:1996 [IEEE/ANSI Std 1003.1 1996 Edition] Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application: Program Interface”, isbn 1-55937-573-6, 1996.

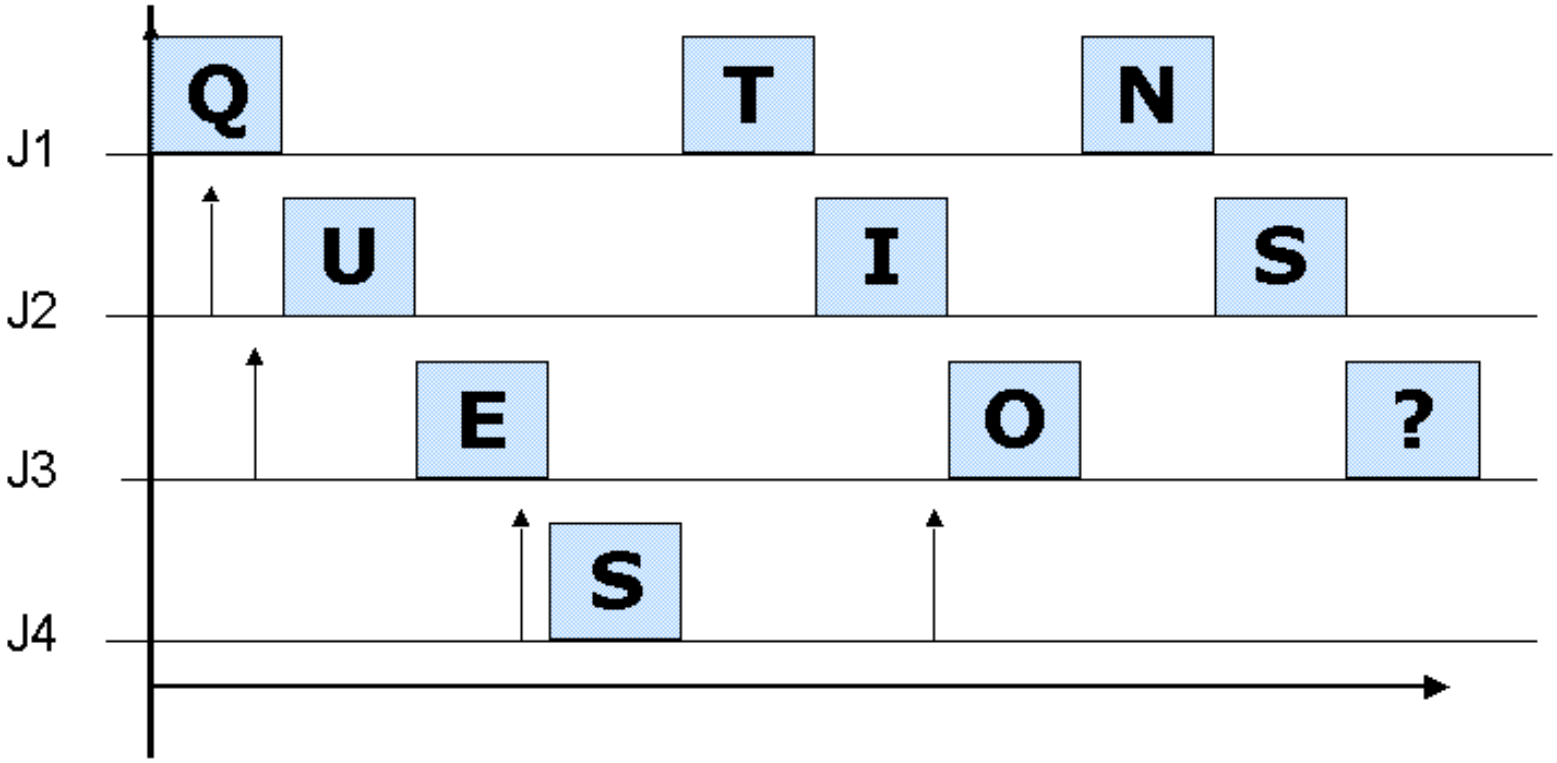
[2] B.O. Gallmeister, “Programming for the Real World - Posix 4”, O’Reilly & Associates, isbn 1-56592-074-0,1995.

[3] J. Migge, A Jean-Marie, N. Navet, “Timing Analysis of Compound Scheduling Policies : Application to Posix1003.1b”, Journal of Scheduling, Kluwer Academic Publishers, vol. 6, n°5, pp457-482, 2003., disponible à l’url http://www.loria.fr/~nnavet.

[4] N. Navet, J. Migge, “Fine Tuning the Scheduling of Tasks through a Genetic Algorithm: Application to Posix1003.1b Compliant OS”, IEE Proceedings Software, IEE, vol. 150, n°1, pp13-24, 2003, disponible à l’url http://www.loria.fr/~nnavet.

[5] R. Brito, N. Navet, “Low-Power Round-Robin Scheduling”, Proc. of the 12th International Conference on Real-Time Systems (RTS’04), Paris, 30-31 march 2004.

[6] F. Jumel, N. Navet, F. Simonot-Lion, “Simulateur d’Architectures Opérationnelles de Contrôle-Commande”, Proc. of the 12th International Conference on Real-Time Systems (RTS’04), Paris, 30-31 march 2004.


J1
J2
J3
J4
Q
T
N
U
I
S
E
O
?
S