Scheduling under Posix 1003.1b

Nicolas NAVET

INRIA - LORIA
TRIO research group
http://www.loria.fr/~nnavet

Posix1003.1b : a standard for real-time Unixes

- **POSIX :** family of IEEE/ISO standards (IEEE committee number 1003) aimed at standardizing services and interfaces offered by Unix-like OS. Eg, Posix 1003.1 (C library), Posix 1003.2 (Shell), **Posix 1003.1b (real-time)**, Posix 1003.1c (threads)

- **Benefit of Posix** : portability at the source code level

- **History :** first draft in 1985, Posix 1003.1 in 1990, 1003.1b in 1993 (aka “posix.4”), 1003.1c in 1995, second version of Posix 1003.1b in 1996

- **Posix 1003.1b defines “real-time” features :** semaphores, shared memory, locking processes in RAM, memory-mapped files, asynchronous I/O, high-res clocks and timers, signals, **scheduling**

Posix 1003.1b : scheduling policies (1/2)

✔ Posix 1003.1b defines 3 policies **SCHED_FIFO, SCHED_OTHER, SCHED_RR**

SCHED_FIFO :

- Priority base preemptive scheduling (at least 32 priority levels)
- FIFO among same-priority tasks
- When
  - the priority of the tasks remains identical over time
  - a priority level is assigned to a single task
    - ⇔ **Fixed Priority Preemptive (FPP)**

SCHED_OTHER :

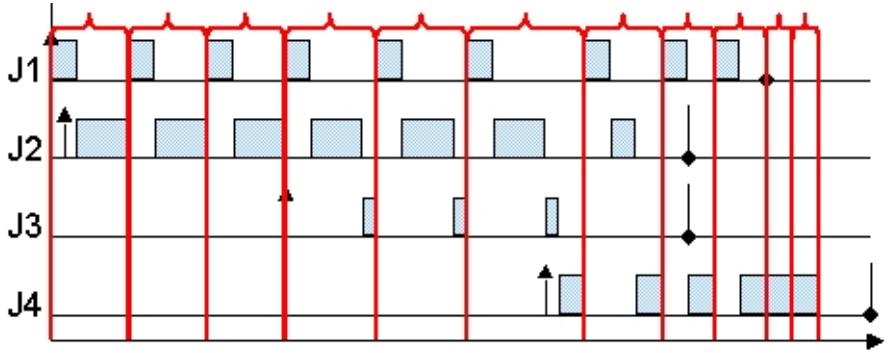
- availability required but not defined by the standard
- usually implements SCHED_FIFO or a classical Unix time-sharing policy

Posix 1003.1b : scheduling policies (2/2)

SCHED_RR : Round-Robin policy

- same priority processes share the CPU
- a process executes during a quantum of time and is then moved at the end of the queue corresponding to its priority level
- higher priority tasks can preempt SCHED_RR tasks
- the size of the quantum can be system-wide (fixed or configurable) or specific for a priority level or a process

✔ **RR-Cycle** : time interval s.t. each active process τ_k uses its quantum Ψ_k (with $\overline{\Psi}_k = \sum \Psi_i - \Psi_k$)



Posix 1003.1b : organization in layers of tasks

✔ Under Posix 1003.1b, **different policies can be used at the same time**

✔ One identifies **layers of tasks** where all tasks inside a layer are scheduled with same policy, layers are prioritized (here $\lambda_1 \succ \lambda_2 \succ \lambda_3$)

Layer	m_l	Tasks	Policy	Priority	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**	-

➠ CPU times allocated to a layer do not depend on the scheduling parameters of the higher priority layers, they only depend on the higher priority workload

Posix 1003.1b : the scheduling API

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

✔ Parameters of the scheduling for a task:

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

✔ Prototypes of the functions:

```
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);
```

Conclusions on the scheduling specification of Posix1003.1b

+ **Actually ease the portability** at the source code level because the specification is simple and widely implemented

+ Offer a **good support for Fixed-Priority Scheduling**

- No EDF scheduling, no periodic task activation service, no Priority Ceiling Protocol

- Priority ranges for the $\neq$ policies are not specified : sometimes distinct priority ranges with SCHED_RR $\leq$ SCHED_FIFO

- Quantum size for RR and the ability to change it is OS-dependent

➠ **Common belief that RR is only useful at the lowest priority levels, it is actually the case ?**

Feasibility of a taskset under Posix 1003.1b (1/2)

- ✔ A system is feasible (or schedulable) if all deadlines are met
- ✔ Feasibility can be evaluated by simulation, feasibility test or by determining response time bounds
- ✔ Context: recurrent tasks with unknown offsets
- ✔ **Response time bounds for a task in a FPP layer:** well known result
  - ➠ Theorem (Lehoczky): under FPP, a bound on the response times can be found for each task in the first busy period after a “critical instant”
  - ➠ Simply add the workload induced by of all higher priority layers

Feasibility of a taskset under Posix 1003.1b (2/2)

✔ **Response time bounds for a task in a RR layer [3]:** an instance $\tau_{k,n}$ can be delayed by

1. the workload of higher priority layers
2. its own work + the workload of the previous instances of the same task : S_k^n leading to $\lceil \frac{S_k^n}{\Psi_k} \rceil$ RR-cycles
3. the time spent for the other tasks of the RR layer :
   - ➠ impossible to know without simulating
   - ➠ conservative assumption : minimum between 1) the sum of the quanta for the other tasks during $\lceil \frac{S_k^n}{\Psi_k} \rceil$ RR-cycles and 2) the overall workload of the other tasks of the layer

✔ In practice, overestimation of around 15% vs actual worst-case response time [4]

✔ For set of tasks with known initial offsets, an exact algorithm exists [5]

Critical Sections and Priority Ceiling Protocol under Posix 1003.1b

---

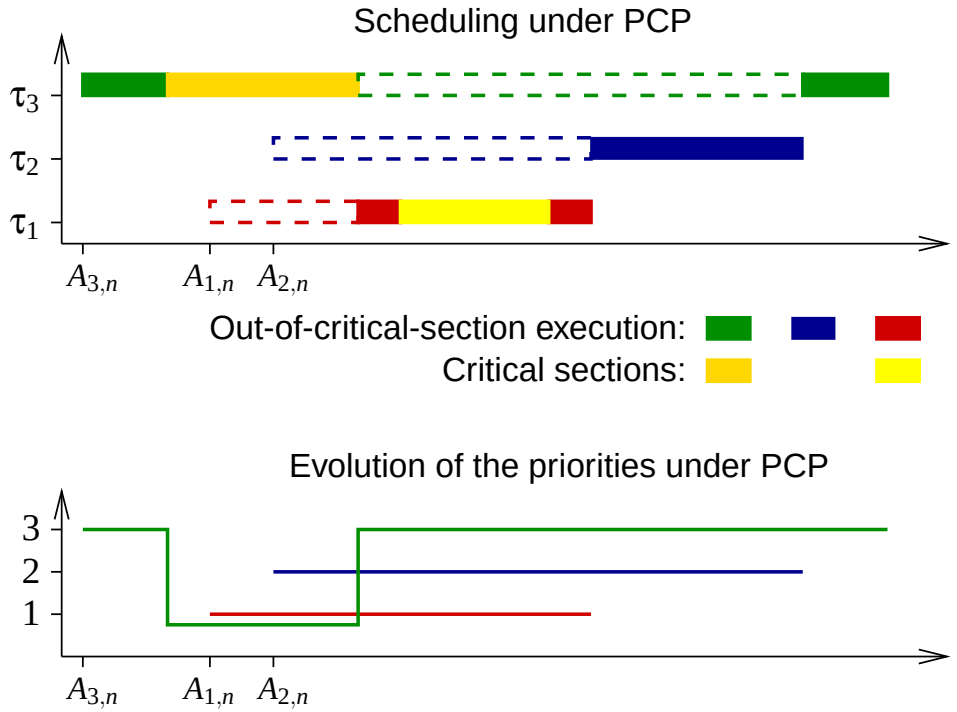
1. **Reminder: PCP under FPP**

2. **PCP under RR : problems and solutions**

Priority Ceiling Protocol (PCP) under FPP

- **PCP** : when a task enters a critical section, it takes the priority of the highest priority task that can use the resource

- 3 tasks τ_1, τ_2 and τ_3, a resource is shared between τ_1 and τ_3 with $\tau_1 \succ \tau_2 \succ \tau_3$



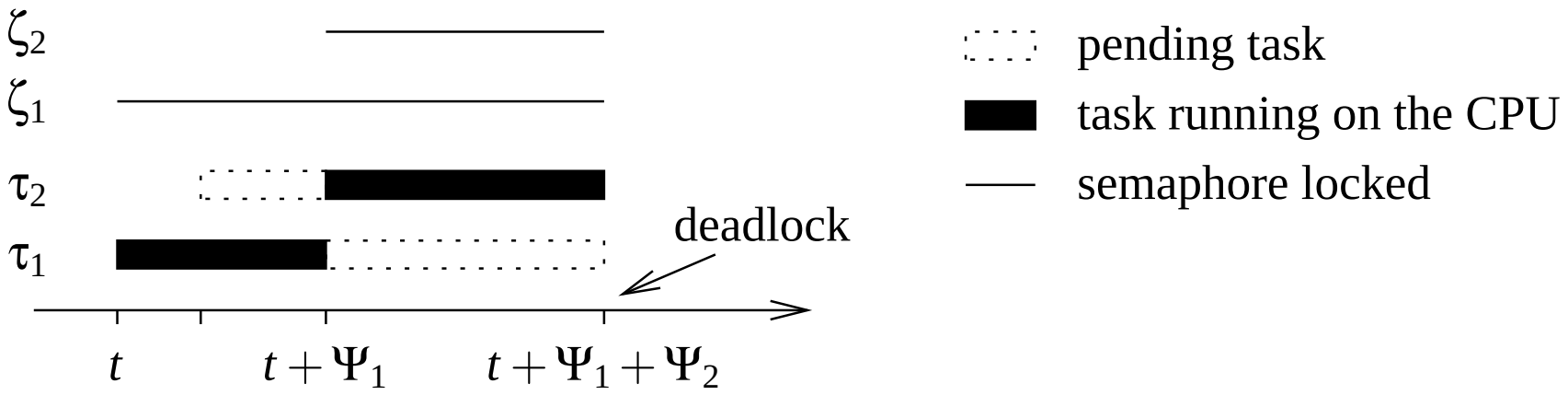
PCP under RR : the problem

Observation 1 : PCP as no effect for RR tasks having the same priority level

Observation 2 : semaphores alone lead to deadlocks under RR

τ_1 needs ζ_1 then needs ζ_2

τ_2 needs ζ_2 then needs ζ_1



PCP under RR : a first solution

Proposition 1 (PCP-RR.1) [3]: PCP + the time quantum of a task is ignored while it executes a critical section

+ Easy to implement, when entering a critical section

1. raise the priority of the task to the ceiling of the resource (PCP)
2. set the quantum value so that the task can finish the critical section

- Distortion with respect to RR fairness

- Worst-case response time : during each RR cycle, a task τ_k can use up to $\Psi_k + z_k$ time units where Ψ_k is the quantum size and z_k the size of the longest critical section of τ_k

➠ **PCP-RR.1 is fine if the size of the critical sections are small compared to the quantum size**

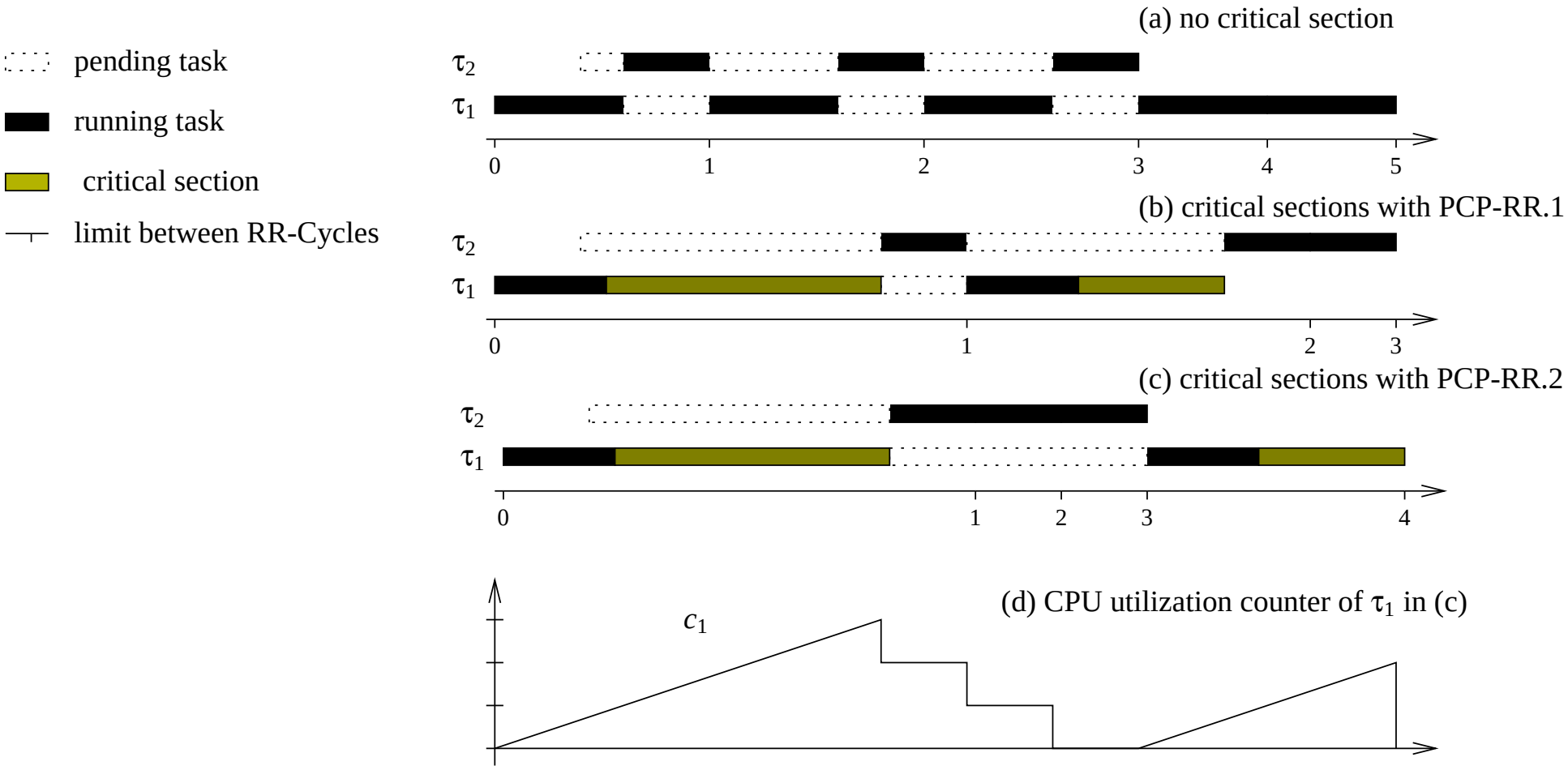
PCP under RR : a better solution

Proposition 2 (PCP-RR.2) [3]: PCP-RR.1 + “reimbursement” of the CPU time received in excess. A processor utilization counter c_k is incremented while the task executes

1. **Task selection :** let τ_k be the next task in the RR cycle
   (a) if $c_k < \Psi_k$ then the oldest instance of τ_k is chosen
   (b) if $c_k \geq \Psi_k$ then $c_k := c_k - \Psi_k$ and τ_k is not chosen

2. **Suspension of the running task :** instance $\tau_{k,n}$ is executed
   (a) $\tau_{k,n}$ in critical section when c_k becomes larger than Ψ_k then $\tau_{k,n}$ is interrupted just after the critical section $c_k := c_k - \Psi_k$
   (b) $\tau_{k,n}$ not in a critical section when c_k becomes larger than Ψ_k then $\tau_{k,n}$ is interrupted $c_k = 0$
   (c) $c_k \leq \Psi_k$ when $\tau_{k,n}$ finishes and $\tau_{k,n+1}$ not ready then $c_k = 0$

PCP under RR : an example



➠ Under PCP-RR.2, at any time, a task τ_k cannot have received more than z_k CPU time units compared to RR without critical sections

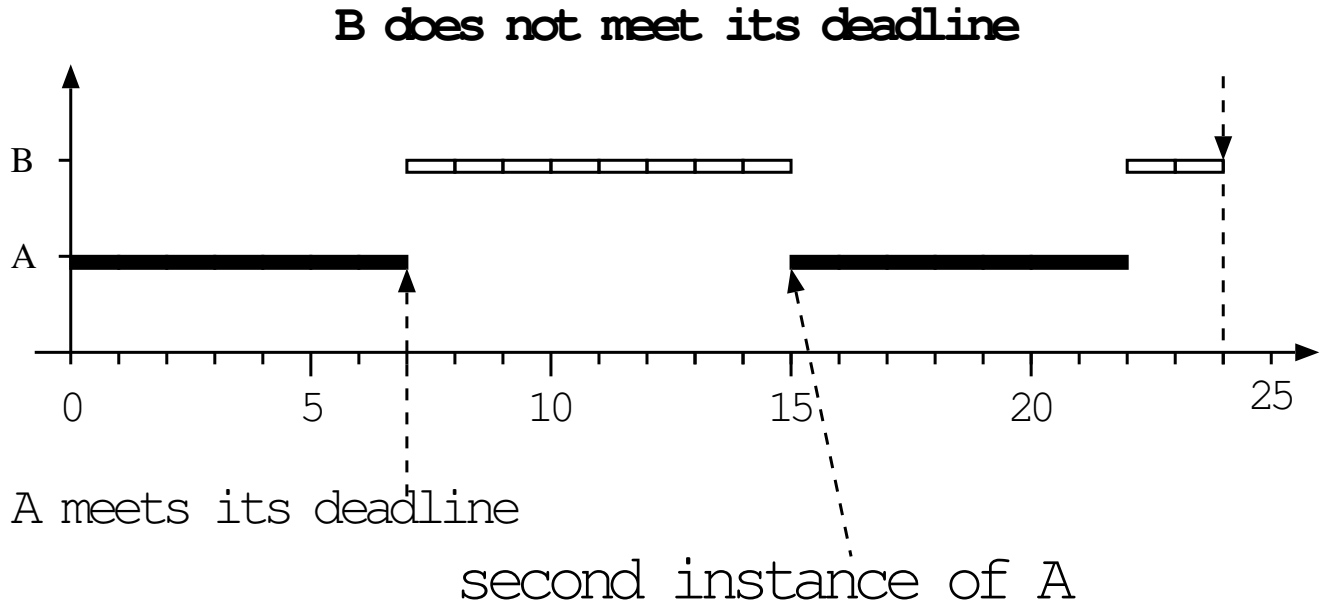
Interest of RR for real-time computing

1. **RR improves the schedulability**

2. **RR helps to satisfy application dependent criteria**

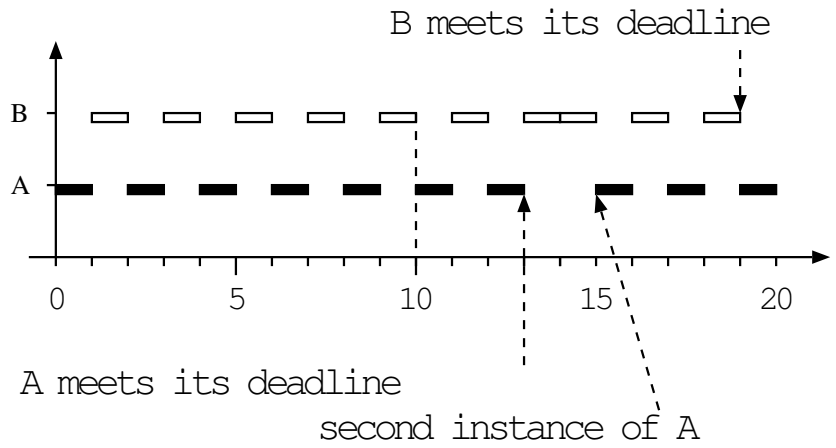
RR and schedulability (1/3)

- ✔ Task A $(C = 7, \overline{D} = 15, T = 15)$ - task B $(C = 10, \overline{D} = 20, T = 50)$
- ✔ Under FPP, Deadline Monotonic (DM) is optimal when $\overline{D} \leq T$

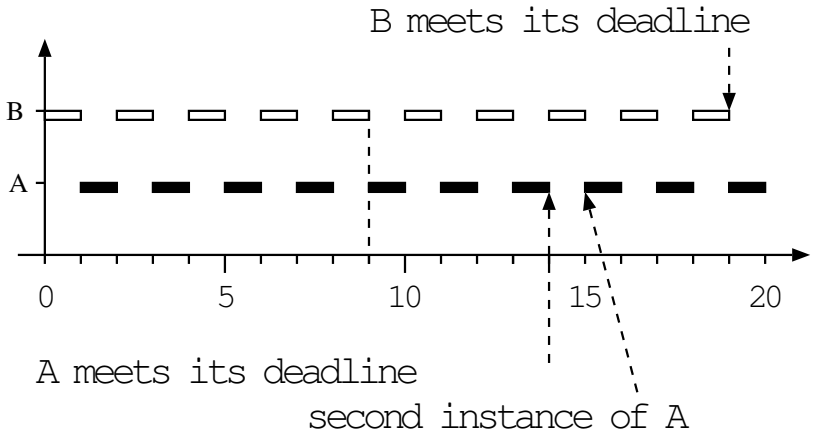


RR and schedulability (2/3)

RR scheduling 1



RR scheduling 2



RR and schedulability (3/3)

✔ Choosing priorities and policies is a combinatorial problem:

- partition of a set of n tasks into k priority levels (=non-empty subsets) : $\frac{1}{k!}\sum_{i=0}^{k}(-1)^{(k-i)}\binom{k}{i}i^n$
- policy is induced by the cardinality of the subset : $\# = 1$ implies FPP, $\# > 1$ implies RR
- there are $k!$ possibility to prioritize the k subsets
- k ranges from 1 to n

Overall there are $\sum_{k=1}^{n}\sum_{i=0}^{k}(-1)^{(k-i)}\binom{k}{i}i^n$ possibilities to allocate priorities and policies (10^8 for 10 tasks)

Experiments 1 [4] : a “random” search with 2000 tries enables, using RR+FPP, to schedule 15% of tasksets which are non-feasible with FPP alone

➠ Good heuristics or even an optimal algorithm (e.g. Branch & Bound) could be found ...

Satisfaction of application dependent criteria

✔ Typically, for computer based control systems, one has to

1. minimize the task end-of-execution jitters
2. maximize the freshness of input data used by the control algorithm
3. maximize the temporal consistency of input data by the control algorithm

Experiments 2 [4]: search with a GA - improvements with FPP+RR compared to FPP alone

results: criterion 1: better in 36% - criterion 2: better in 68% - criterion 3: better in 70%

Experiments 3 [6]: multi-tasks with 3 control laws - simulation with mathlab/simulink - criteria: response time and overshoot - improvements with RR alone compared to FPP alone

results: RR significantly better than FPP but outperformed by other policies

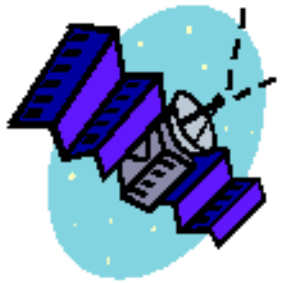
Scheduling under time and energy constraints

Context:

GSM

Ordinateurs portables

Implants medicaux



satellites

1. **Taxonomy of existing solutions for RR and FPP**

2. **Existing work for RR**

Existing solutions

Low-power scheduling : **use the smallest frequency that ensures the respect of performance constraints**

Techniques and existing work : still nothing for RR+FPP !

1. **Single static frequency for the whole system: FPP and RR**
2. **Single frequency for task : FPP**
3. **Single frequency for instance : FPP**
4. **Frequency chosen for each CPU instruction : FPP**
5. **Redistribution of unused CPU times** (with or without code instrumentation) : **FPP**

Single static frequency under RR

Result 1: the frequency under any policy is larger than or equal to the frequency under (S_{edf})

Algorithm (periodic tasks): **starting from S_{edf}, choose the smallest available frequency s.t. the bound on the response times is lower than the deadline for each task**

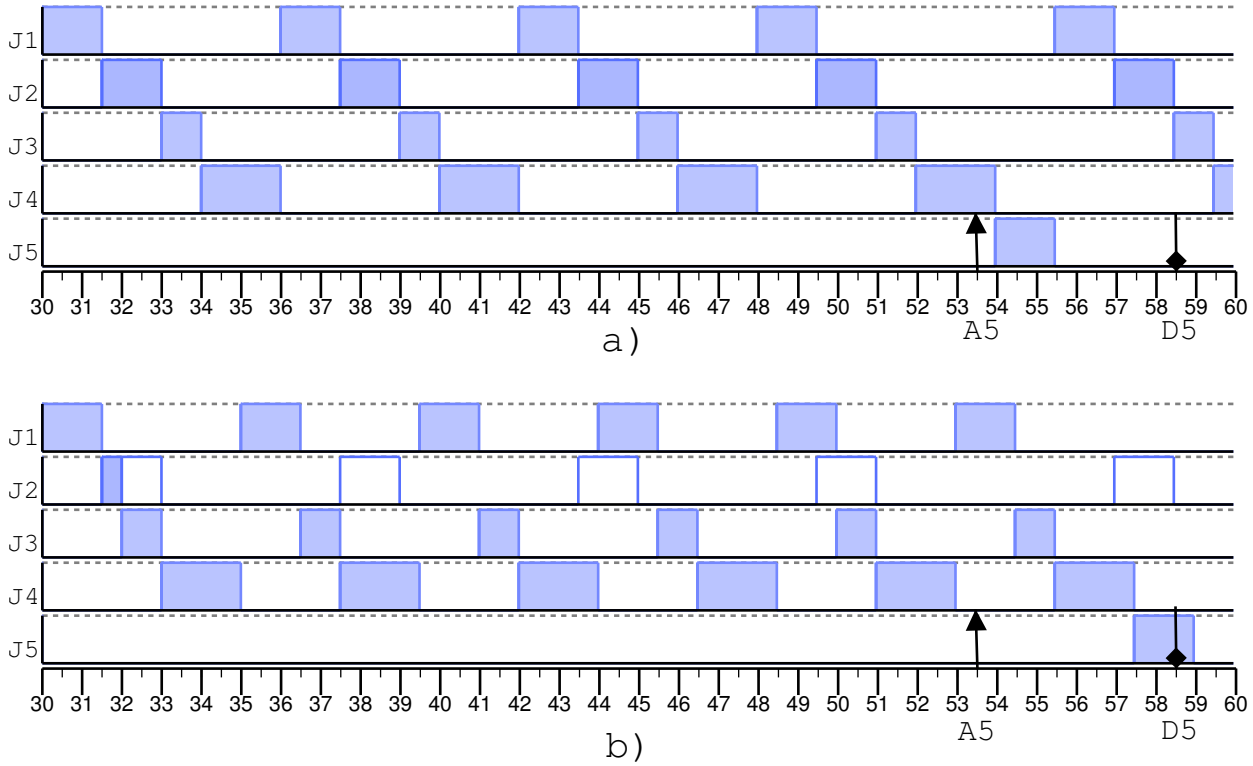
➠ **Pb :** energy reduction is sub-optimal since the computation of the response time bounds is pessimistic under RR

✔ There is an [5] optimal algorithm for periodic tasks in a single RR layer with known initial offsets, one observes that:

1. a job that terminates earlier than planned can lead to the unfeasibility of the system
2. sometimes a system is feasible with a lower frequency and not feasible with a higher one

Low-power RR scheduling (1/2)

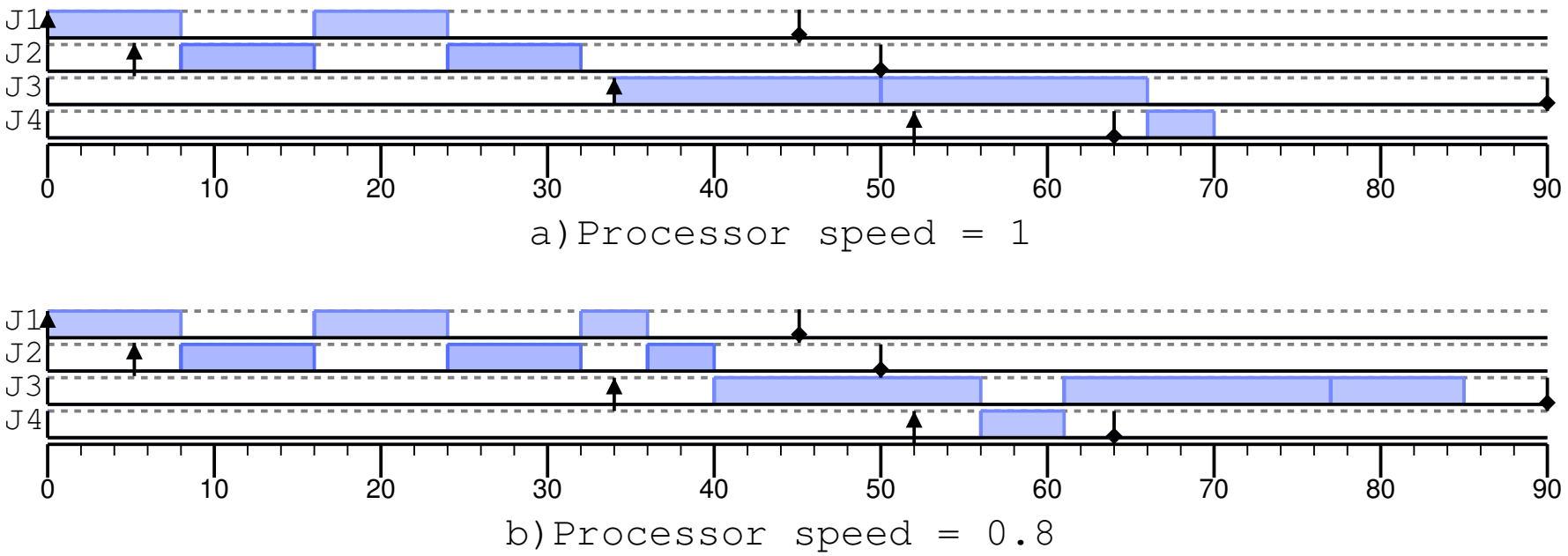
Observation 1: The termination of a job earlier than its expected WCET can lead to the unfeasibility of the system



✔ On-line mechanisms exist [5] for preventing this phenomenon ..

Low-power RR scheduling (2/2)

Observation 2: Under RR, feasibility is not monotonous in the frequency of the CPU



✔ Start from S_{edf} and choose the lowest admissible frequency ..

Conclusions / Future Work

- ✔ Posix 1003.1b a good support for Fixed-Priority Scheduling
- ✔ but does not give any guarantee on the performances of the OS (eg: preemptible kernel calls)
- ✔ RR is not well defined and not considered suitable for “important” tasks

Perspectives:

- ➠ Audsley-like algorithm for setting priorities and policies of tasks with fixed-sized and variable-size quanta (near-EDF feasibility ?)
- ➠ algorithms and mechanisms for power-aware real-time scheduling

Software

Downloadable at url http://www.loria.fr/~nnavet :

- ✔ **RTS et TkRTS** written by J. Migge for computing response time bounds under FPP, NP-FPP, EDF, NP-EDF, RR

- ✔ **a scheduling simulator** (JAVA applet Java with chronogram) for all time-independent policy (i.e. priority of an instance fixed over time)

- ✔ **Applicative level scheduler** with a periodic task activation service that can be configured for all time independent policy

References

[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.

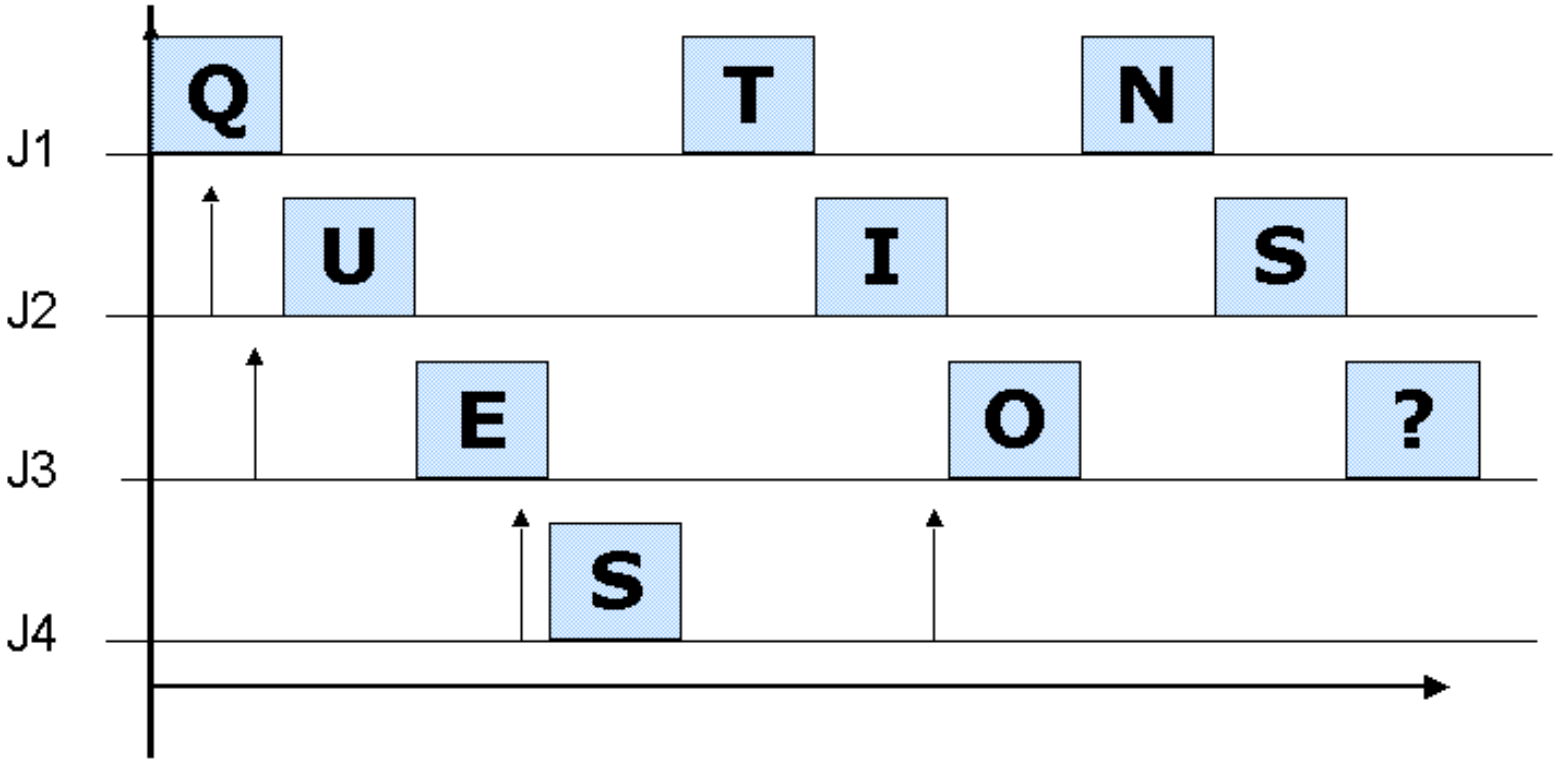
[2] B.O. Gallmeister, “Programming for the Real World - Posix 4”, O’Reilly & Associates, isbn 1-56592-074-0,1995.

[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., available at 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, available at 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, available at url http://www.loria.fr/~nnavet.

[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