Building Embedded Real-Time Applications
Part 2 : Real-time scheduling theory and Ada Real-time programming

John Mc Cormick, Frank Singhoff

mccormick@cs.uni.edu, singhoff@univ-brest.fr

Talk overview

1. **Introduction**
   - Real-time systems.
   - What is real-time scheduling.
   - What we aim to do this afternoon ?
2. **Real-time scheduling theory**
   - Introducing real-time scheduling theory.
   - Usual real-time schedulers.
   - Few words about shared resources.
3. **Ada standards and real-time scheduling**
   - Real-time scheduling with Ada 1995/2005, Ravenscar.
   - POSIX 1003 and its Ada binding.
4. **Scheduling analysis tools and how to use them.**
5. **Summary and further readings**

Real-time systems

- "**Real-time systems** are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but **also on the time** at which the results are produced" [STA 88].

- **Properties we look for** :
  - **Functions must be predictable** : the same data input will produce the same data output.
  - **Timing behavior must be predictable** : must meet temporal constraints (e.g. deadline, response time).

$\Longrightarrow$ Predictable means ... we can **compute** the system temporal behavior **before** execution time.

What is real-time scheduling theory (1)

- **Many real-time systems are built with operating systems providing multitasking facilities, in order to:**
  - Ease the design of complex systems (one function = one task).
  - Increase efficiency (I/O operations, multi-processor architecture).
  - Increase re-usability.

What is real-time scheduling theory (1)

- **Many real-time systems are built with operating systems providing multitasking facilities, in order to:**
  - Ease the design of complex systems (one function = one task).
  - Increase efficiency (I/O operations, multi-processor architecture).
  - Increase re-usability.

But, multitasking makes the predictability analysis difficult to do : we must take the task scheduling into account in order to check temporal constraints $\Longrightarrow$ schedulability analysis.

What is real-time scheduling theory (2)

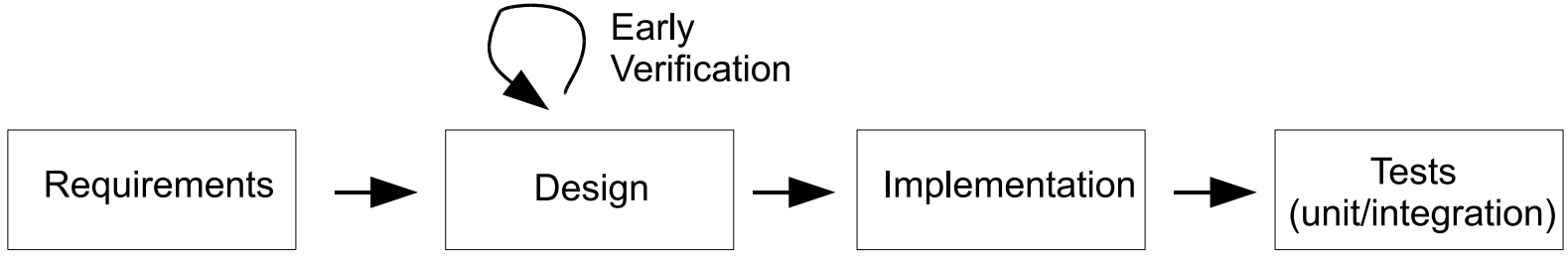
- **Example of a software embedded into a car:**

1. Tasks are released several times and have a job to do for each release.
2. Each task completes its current job before it has been released for the next one.
3. A task displays every 100 milliseconds the current speed of the car.
4. A task reads a speed sensor every 250 milliseconds.
5. A task performs an engine monitoring algorithm every 500 milliseconds.

$\Longrightarrow$ How can we check that tasks will be ran at the proper rate? Do they meet their timing requirements? Do we have enough processor resource?

$\Longrightarrow$ If the system is complex (eg. large number of tasks), the designer must be helped to perform such an analysis.

What is real-time scheduling theory (3)



- The real-time scheduling theory is a framework which provides :

1. **Algorithms to share a processor** (or any resources) by a set of tasks (or any resource users) according to some timing requirements $\Longrightarrow$ take urgency of the tasks into account.
2. **Analytical methods,** called **feasibility tests** or **schedulability tests**, which allow a system designer to early analyze/"compute" the system behavior before implementation/execution time.

What we aim to do this afternoon

- **Real-time scheduling theory is born 30 years ago, but few people actually use it:**

  1. This theory is sometimes difficult to be applied (and sometimes unsuitable).
  2. Few analysis tools exist.
  3. Few software designers know such a theory.

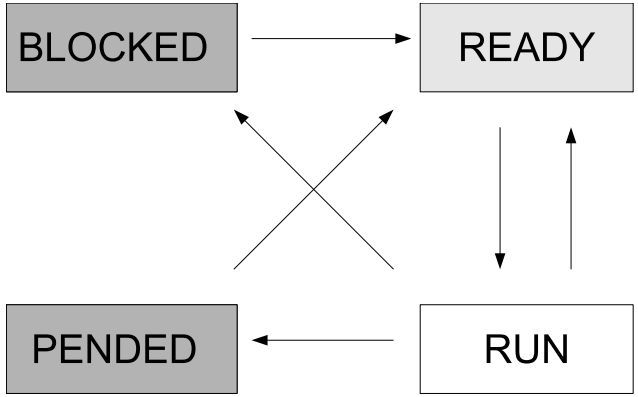
- **This tutorial presents you:**

  1. The foundation of the real-time scheduling theory.
  2. How it can help you to model a real-time application.
  3. How it can help you to analyze a real-time application in order to perform early verification.
  4. How to implement an Ada real-time application which is compliant with real-time scheduling theory assumptions.

Talk overview

1. **Introduction**
   - Real-time systems.
   - What is real-time scheduling.
   - What we aim to do this afternoon ?
2. **Real-time scheduling theory**
   - Introducing real-time scheduling theory.
   - Usual real-time schedulers.
   - Few words about shared resources.
3. **Ada standards and real-time scheduling**
   - Real-time scheduling with Ada 1995/2005, Ravenscar.
   - POSIX 1003 and its Ada binding.
4. **Scheduling analysis tools and how to use them.**
5. **Summary and further readings**

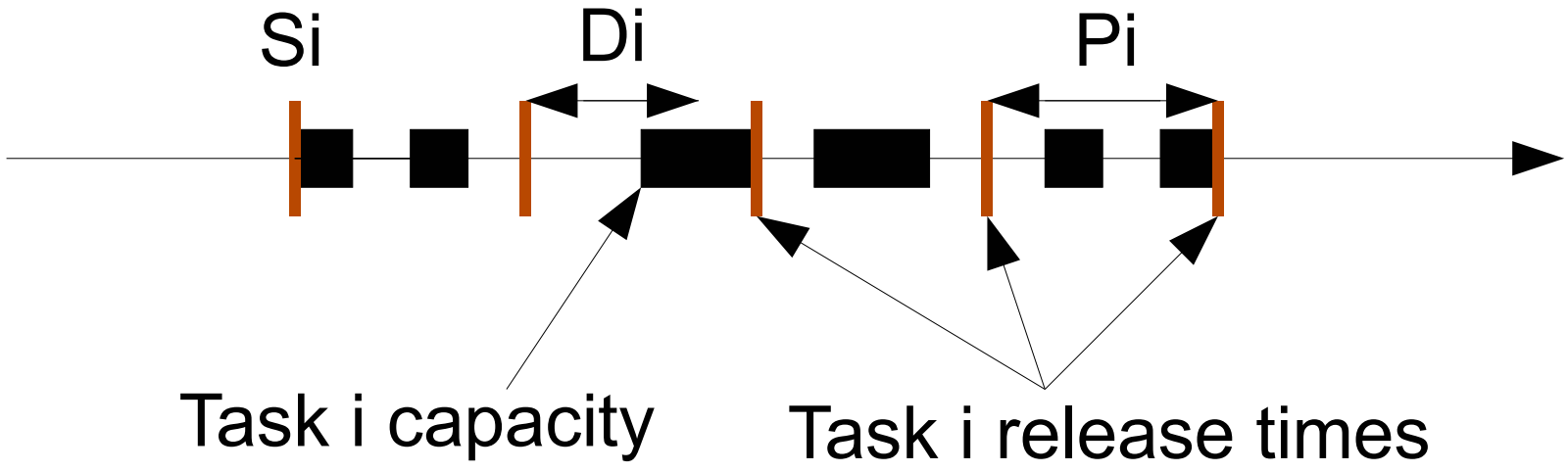
Real-time scheduling theory (1)



- **Task** : sequence of statements + data + execution context (processor and MMU). Usual states of a task.

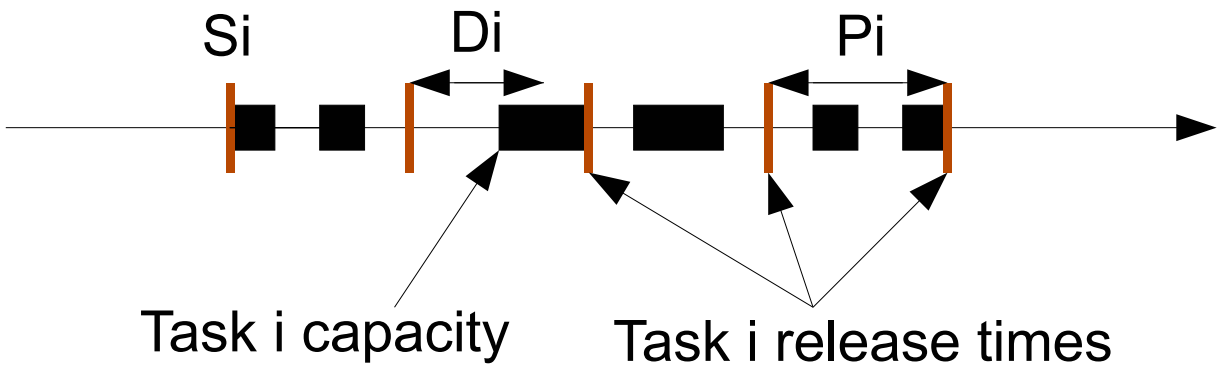
- **Usual task types** :
  - Urgent or/and critical tasks.
  - Independent tasks or dependent tasks.
  - Periodic and sporadic tasks (critical tasks). Aperiodic tasks (non critical tasks).

Real-time scheduling theory (2)



- **Usual parameters of a periodic task i :**
  - First task release time : S_i.
  - Worst case execution time : C_i (or capacity).
  - Period : P_i (duration between two periodic release times).
  - Static deadline to meet : D_i, timing constraint relative to the period.
  - Priority : allows the scheduler to choose the task to run.

Real-time scheduling theory (3)



- **Assumptions for this tutorial (synchronous periodic task with deadlines on requests) [LIU 73]:**

1. All tasks are periodic.
2. All tasks are independent.
3. $\forall i : P_i = D_i$: a task must end its current job before its next release time.
4. $\forall i : S_i = 0 \implies$ called critical instant (worst case on processor demand).

Real-time scheduling theory (4)

- **Different kinds of real-time schedulers:**
  - **On-line/off-line scheduler :** the scheduling is computed before or at execution time ?
  - **Static/dynamic priority scheduler :** priorities may change at execution time ?
  - **Preemptive or non preemptive scheduler :** can we stop a task during its execution ?
    1. Preemptive schedulers are more efficient than non preemptive schedulers (eg. missed deadlines).
    2. Non preemptive schedulers ease the sharing of resources.
    3. Overhead due to context switches.

Real-time scheduling theory (5)

- What we look for when we choose a scheduler :

1. **Feasibility tests (analytical/algebraic tools also called schedulability tests):** can we prove that tasks will meet deadlines before execution time?

2. **Complexity:** can we apply feasibility tests on large systems (eg. large number of tasks) ?

3. **Optimality:** an optimal scheduler is a scheduler which is always able to compute a scheduling if one exists.

4. **Suitability:** can we implement the chosen scheduler in a real-time operating system ?

Usual real-time schedulers

1. **Fixed priority scheduler :** Rate Monotonic priority assignment (sometimes called Rate Monotonic Scheduling or Rate Monotonic Analysis, RM, RMS, RMA).
2. **Dynamic priority scheduler :** Earliest Deadline First (or EDF).

Fixed priority scheduling (1)

- **Assumptions/properties of fixed priority scheduling :**
  - Scheduling based on fixed priority $\Longrightarrow$ static and critical applications.
  - Priorities are assigned at design time (off-line).
  - Efficient and simple feasibility tests.
  - Scheduler easy to implement into real-time operating systems.
- **Assumptions/properties of Rate Monotonic assignment :**
  - Optimal assignment in the case of fixed priority scheduling.
  - Periodic tasks only.

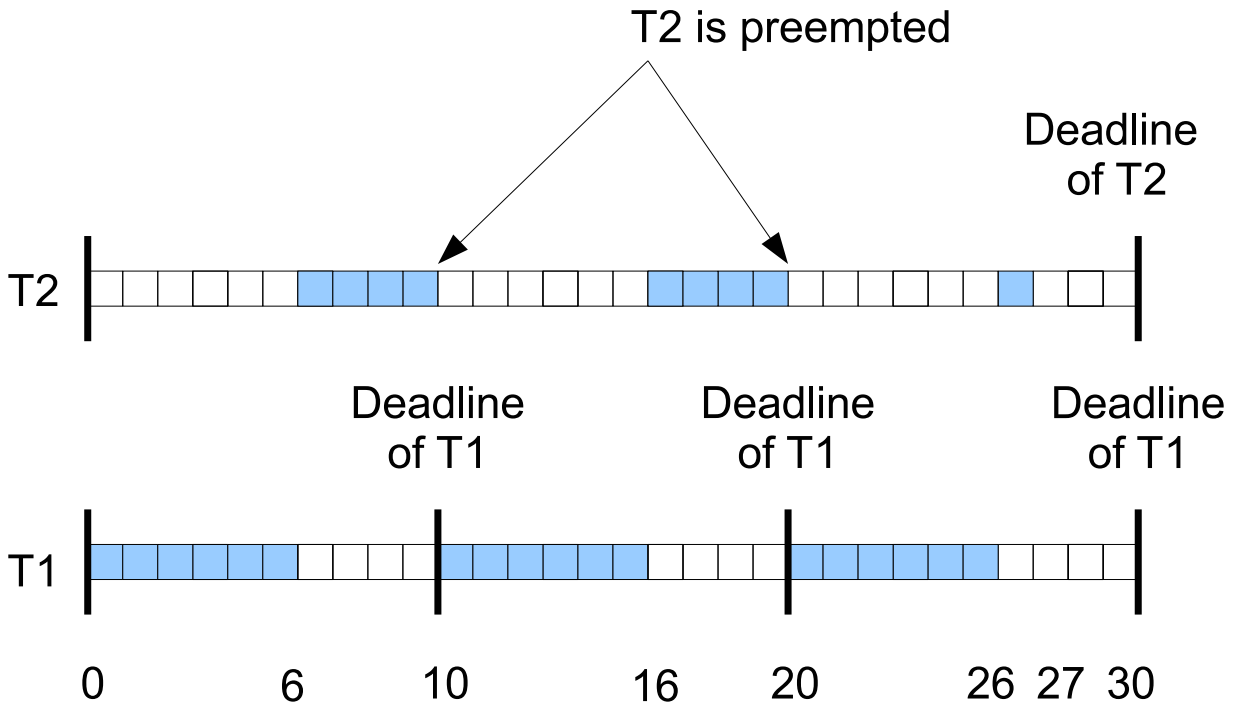
Fixed priority scheduling (2)

- **How does it work :**

1. **"Rate monotonic" task priority assignment :** the highest priority tasks have the smallest periods. Priorities are assigned off-line (e.g. at design time, before execution).
2. **Fixed priority scheduling :** at any time, run the ready task which has the highest priority level.

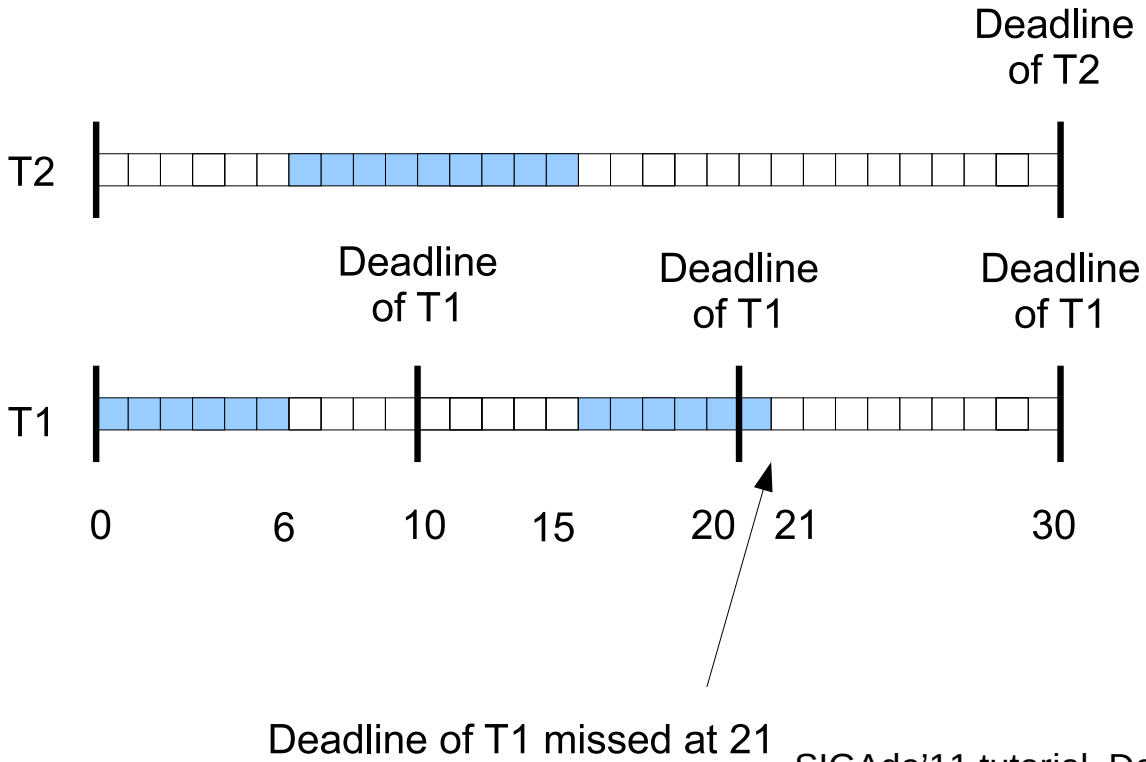
Fixed priority scheduling (3)

- Rate Monotonic assignment and preemptive scheduling:
  - Assuming VxWorks priority levels (high=0 ; low=255)
  - T1 : C1=6, P1=10, Prio1=0
  - T2 : C2=9, P2=30, Prio2=1



Fixed priority scheduling (4)

- Rate Monotonic assignment and non preemptive scheduling:
  - Assuming VxWorks priority levels (high=0 ; low=255)
  - T1 : C1=6, P1=10, Prio1=0
  - T2 : C2=9, P2=30, Prio2=1



Fixed priority scheduling (5)

- **Feasibility/Schedulability tests :**

1. **Run simulations on scheduling period** = $[0, LCM(P_i)]$. Sufficient and necessary (exact result). Any priority assignment and preemptive/non preemptive scheduling.

2. **Processor utilization factor test** :

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1)$$

   Rate Monotonic assignment and preemptive scheduling. Sufficient but not necessary. Does not compute an exact result.

3. **Task worst case response time, noted** $r_i \implies$ delay between task release time and task end time. Can compute an exact result. Any priority assignment and preemptive scheduling.

Fixed priority scheduling (6)

- **Compute r_i, the task worst case response time :**
  - Assumptions : preemptive scheduling, synchronous periodic tasks.
  - task i response time $=$ task i capacity + delay the task i has to wait due to higher priority task j. Or :

$$r_i = C_i + \sum_{j \in hp(i)} WaitingTime_j$$

or:

$$r_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{P_j} \right\rceil C_j$$

  - $hp(i)$ is the set of tasks which have a higher priority than task i. $\lceil x \rceil$ returns the smallest integer not smaller than x.

Fixed priority scheduling (7)

- **To compute task response time** : compute w_i^k with:

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{P_j} \right\rceil C_j$$

- Start with $w_i^0 = C_i$.
- Compute w_i^1, w_i^2, w_i^3, ... w_i^k upto:
  - If $w_i^k > P_i$. No task response time can be computed for task i. Deadlines will be missed !
  - If $w_i^k = w_i^{k-1}$. w_i^k is the task i response time. Deadlines will be met.

Fixed priority scheduling (8)

- **Example:** T1 (P1=7, C1=3), T2 (P2=12, C2=2), T3 (P3=20, C3=5)

  - $w_1^0 = C1 = 3 \Longrightarrow r_1 = 3$
  - $w_2^0 = C2 = 2$
  - $w_2^1 = C2 + \left\lceil \frac{w_2^0}{P1} \right\rceil C1 = 2 + \lceil \frac{2}{7} \rceil 3 = 5$
  - $w_2^2 = C2 + \left\lceil \frac{w_2^1}{P1} \right\rceil C1 = 2 + \lceil \frac{5}{7} \rceil 3 = 5 \Longrightarrow r_2 = 5$
  - $w_3^0 = C3 = 5$
  - $w_3^1 = C3 + \left\lceil \frac{w_3^0}{P1} \right\rceil C1 + \left\lceil \frac{w_3^0}{P2} \right\rceil C2 = 5 + \lceil \frac{5}{7} \rceil 3 + \lceil \frac{5}{12} \rceil 2 = 10$
  - $w_3^2 = 5 + \lceil \frac{10}{7} \rceil 3 + \lceil \frac{10}{12} \rceil 2 = 13$
  - $w_3^3 = 5 + \lceil \frac{13}{7} \rceil 3 + \lceil \frac{13}{12} \rceil 2 = 15$
  - $w_3^4 = 5 + \lceil \frac{15}{7} \rceil 3 + \lceil \frac{15}{12} \rceil 2 = 18$
  - $w_3^5 = 5 + \lceil \frac{18}{7} \rceil 3 + \lceil \frac{18}{12} \rceil 2 = 18 \Longrightarrow r_3 = 18$

Fixed priority scheduling (9)

- **Analysis of the car with embedded software example:**

1. $Tdisplay$: task which displays speed. P=100, C=20.
2. $Tspeed$: task which reads speed sensor. P=250, C=50.
3. $Tengine$: task which runs an engine monitoring program. P=500, C=150.

- **Processor utilization test:**

$U = \sum_{i=1}^{n} \frac{C_i}{P_i} = 20/100 + 150/500 + 50/250 = 0.7$

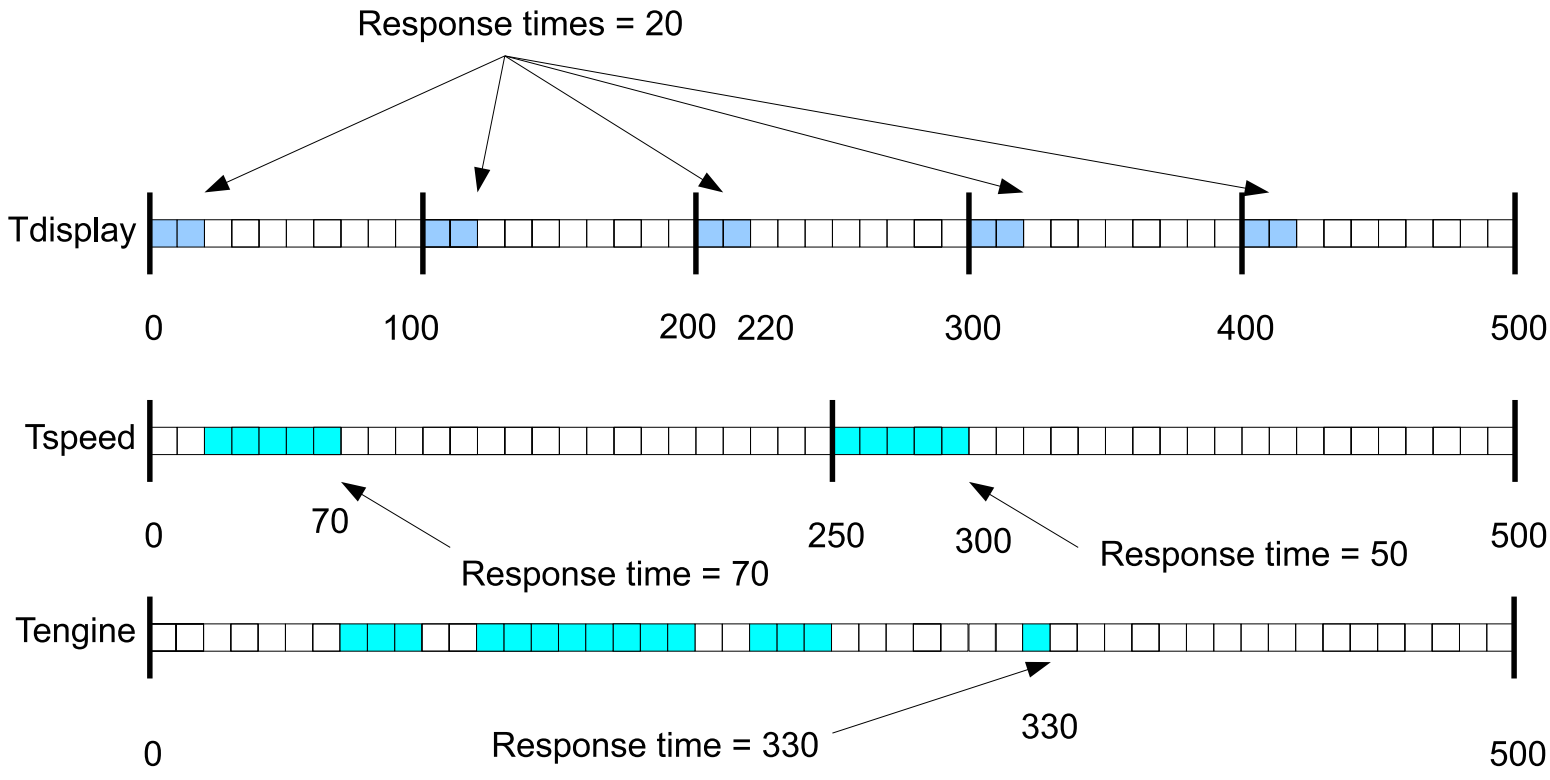
$Bound = n(2^{\frac{1}{n}} - 1) = 3(2^{\frac{1}{3}} - 1) = 0.779$

$U \leq Bound \Longrightarrow$ deadlines will be met.

- **Task response time:** $r_{Tengine} = 330$, $r_{Tdisplay} = 20$, $r_{Tspeed} = 70$.

Fixed priority scheduling (10)

- **... and check on the computed scheduling**



- Run simulations on **scheduling period** = $[0, LCM(P_i)]$ = $[0, 500]$.

Earliest Deadline First (1)

- **Assumptions and properties:**
  - Dynamic priority scheduler $\Longrightarrow$ suitable for dynamic real-time systems.
  - Is able to schedule both aperiodic and periodic tasks.
  - Optimal scheduler : can reach 100 % of the cpu usage.
  - But difficult to implement into a real-time operating system.
  - Becomes unpredictable if the processor is over-loaded $\Longrightarrow$ not suitable for hard-critical real-time systems.

Earliest Deadline First (2)

- **How does it work :**

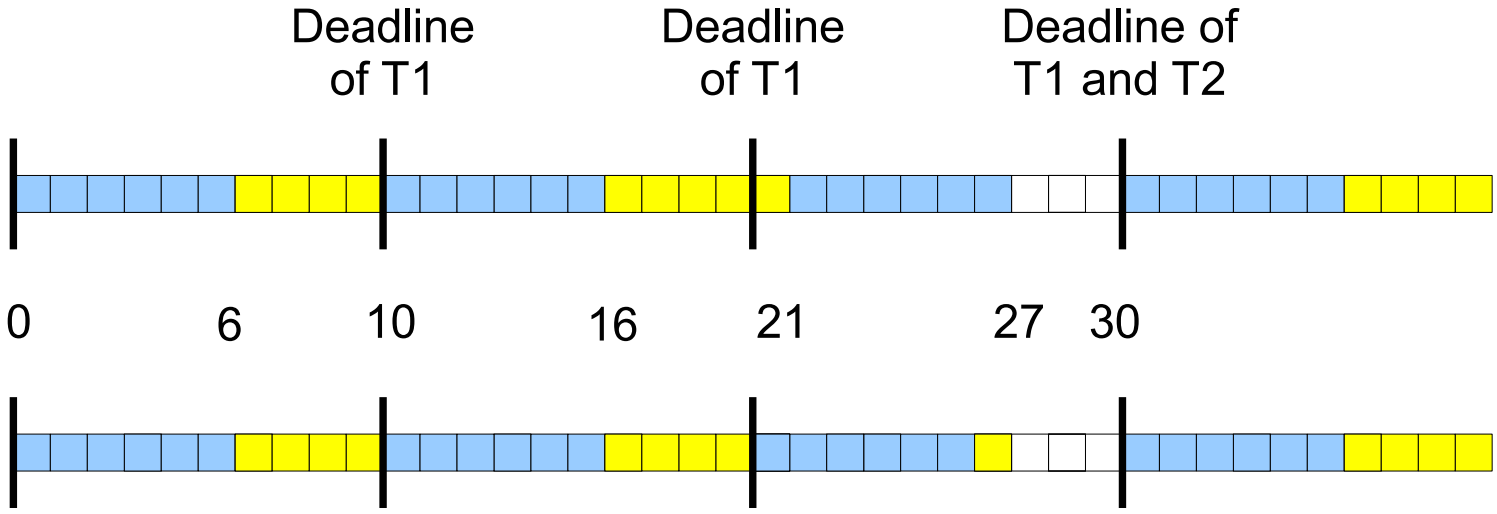
1. **Compute task priorities (called "dynamic deadlines")** $\Longrightarrow D_i(t)$ is the priority/dynamic deadline of task i at time t:
   - Aperiodic task : $D_i(t)$ = D_i + S_i.
   - Periodic task : $D_i(t)$ = k + D_i, where k is the task release time before t.

2. **Select the task :** at any time, run the ready task which has the shortest dynamic deadline.

Earliest Deadline First (3)

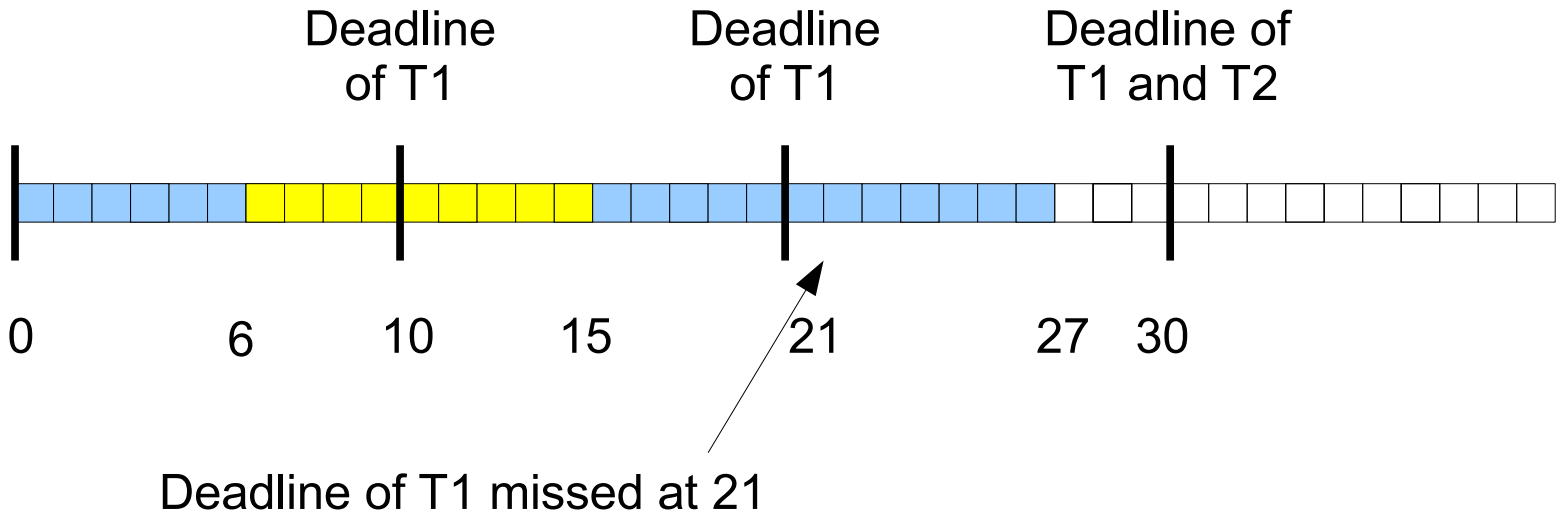
- Preemptive case: (T1/blue, T2/yellow, C1=6; P1=10; C2=9; P2=30)

t	$D_1(t)$	$D_2(t)$
0..9	$k + D_1 = 0 + 10 = 10$	$k + D_2 = 0 + 30 = 30$
10..19	$k + D_1 = 10 + 10 = 20$	30
20..29	$k + D_1 = 20 + 10 = 30$	30



Earliest Deadline First (4)

- Non preemptive case:



Earliest Deadline First (5)

- **Feasibility tests/schedulability tests :**

1. Run simulations on **scheduling period** = $[0, LCM(P_i)]$. Sufficient and necessary (exact result).

2. **Processor utilization factor test** (eg. preemptive case) :

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

   Sufficient and necessary. Difficult to use in real life applications. Compute an exact result.

3. **Task response time :** a bit more complex to compute (dynamic scheduler) !

EDF vs FP/RM : summary [BUT 03]

- Aperiodic task = non critical task.
- Periodic task = critical task.

	FP/RM	EDF
Applications	critical, static	dynamic, less critical
RTOS implementation	easy	more difficult
Tasks	Periodic only	Aperiodic and periodic
Efficiency	upto 69 %	upto 100 %
Predictability	high	less than FP/RM if $U > 1$

- Warning : the real picture is more complicated: aperiodic tasks scheduling with FP/RM, U=1 with FP/RM, ...

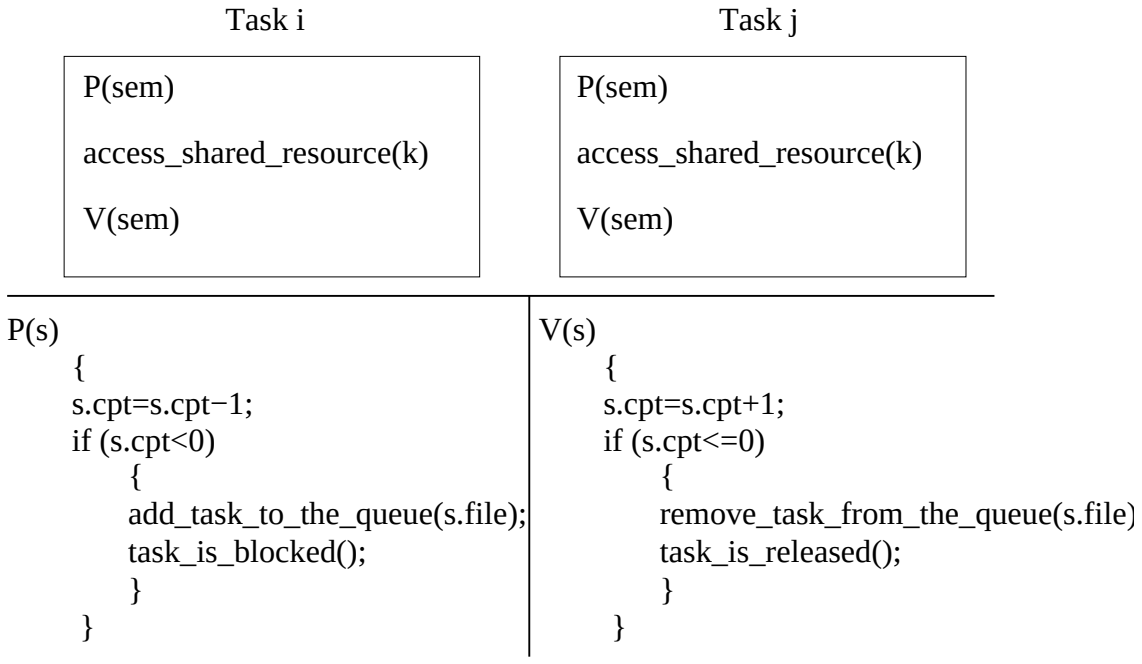
Feasibility tests/scheduler suitability

- **To be applied, algebraic tools and scheduling algorithms that we have presented must be improved to take into account :**

  - The operating system overheads (e.g. task context switches).
  - Usually, tasks are not independent :
    1. Tasks may have precedence constraints (e.g. tasks may exchange messages).
    2. Tasks may share resources (e.g. shared memories).
  - ...

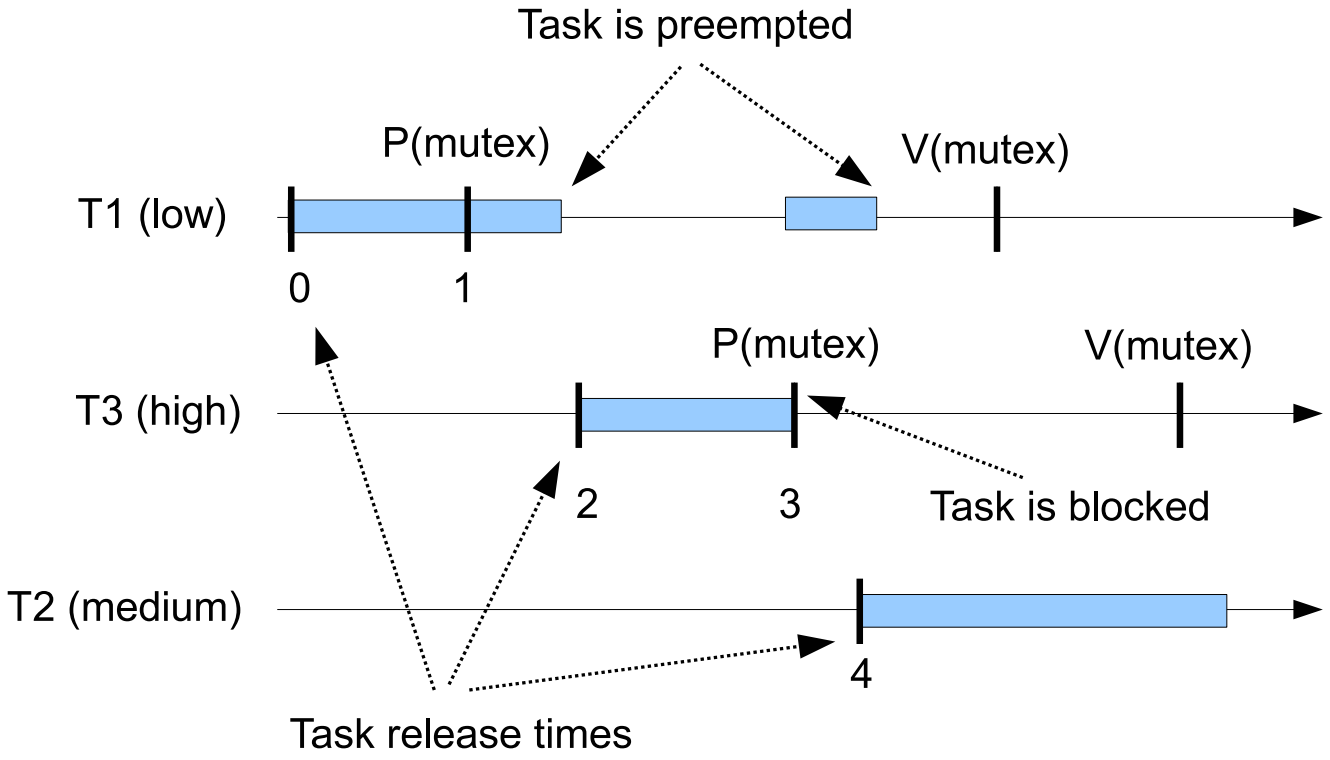
Shared resource support (1)

- **A shared resource may be modeled by a semaphore:**



- Semaphore = counter + a FIFO queue.
- A semaphore may be used to implement critical sections, producers-consumers, blackboards, barriers, ...
- **We use specific semaphores in real-time scheduling systems.**

Shared resource support (2)



- **What is Priority inversion:** a low priority task blocks a high priority task ... allowing a medium priority task to held the processor $\Longrightarrow$ a non critical task runs before a critical task!
- B_i = worst case shared resource waiting time of task i.

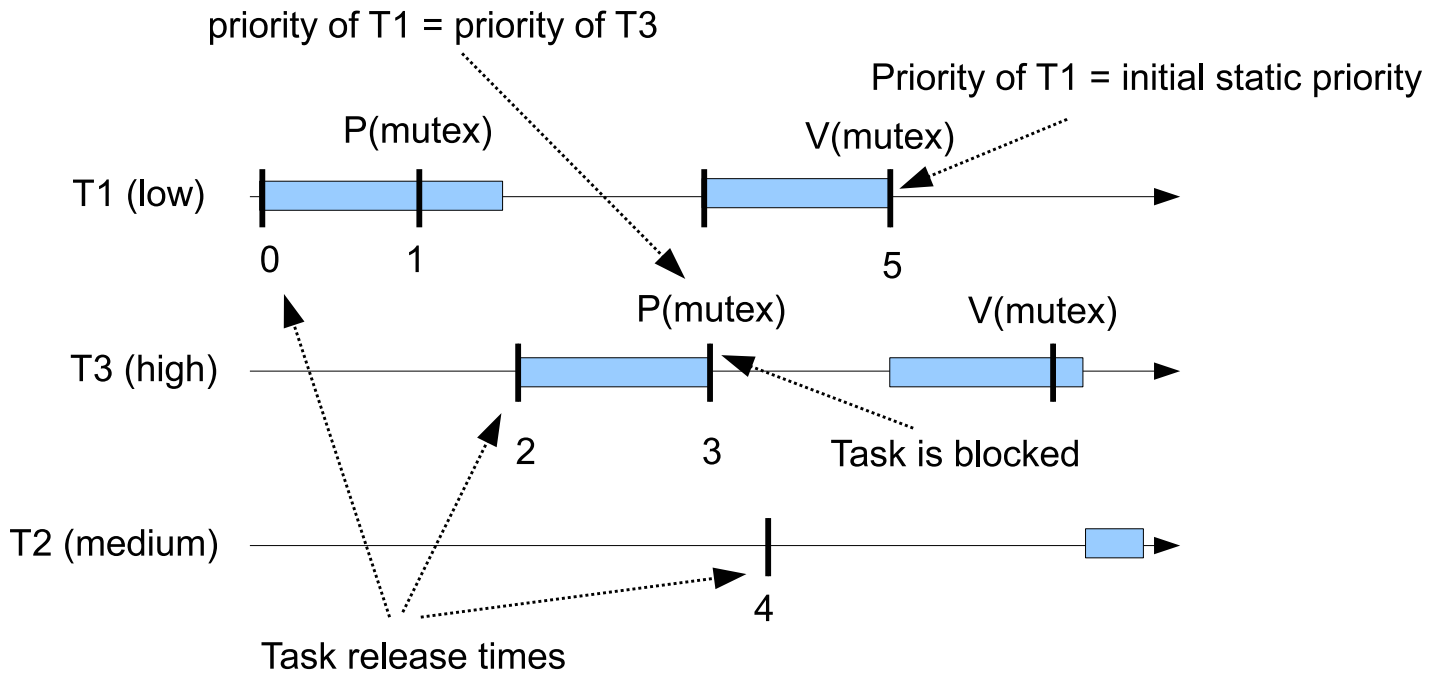
Shared resource support (3)

- How to reduce priority inversion?
- How long a task must wait for the access to a shared resource ? How to compute B_i?

$\Longrightarrow$ **To reduce priority inversion, we use priority inheritance.**

- Priority inheritance protocols provide a specific implementation of $P()$ and $V()$ of semaphores.

Shared resource support (4)

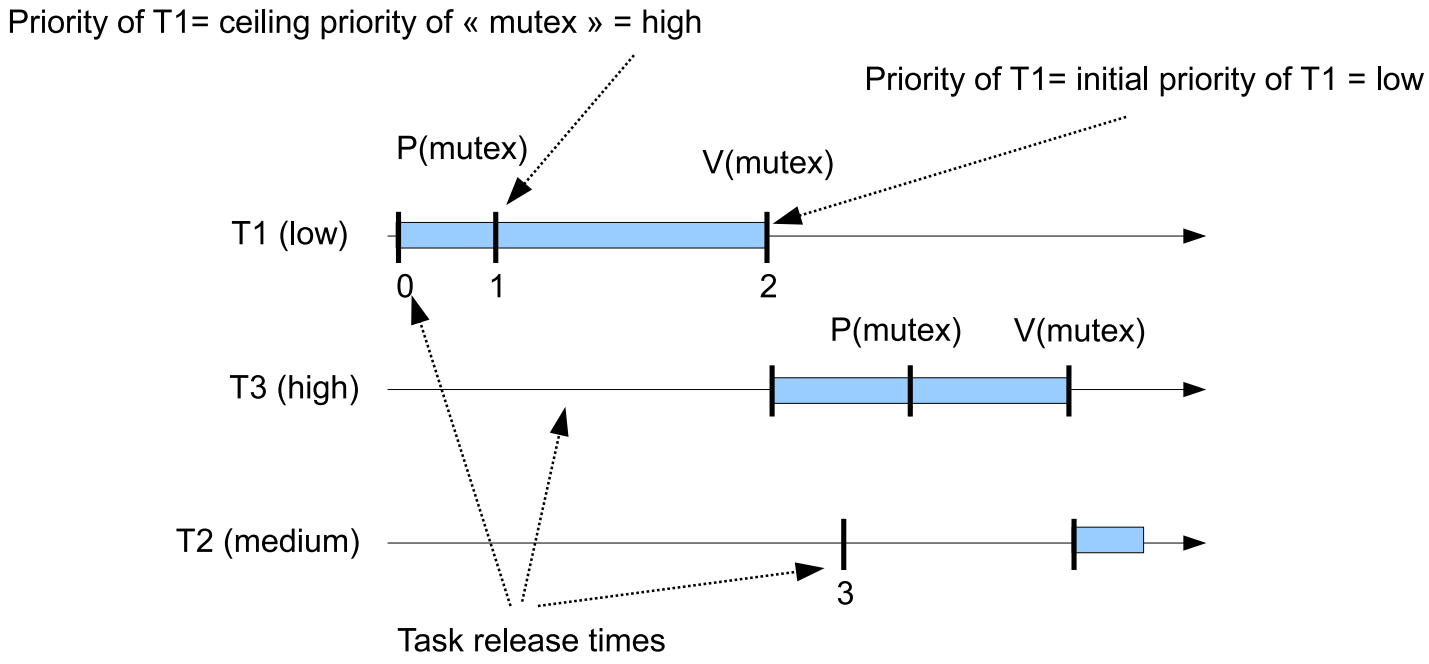


- **Priority Inheritance Protocol or PIP:**
  - A task which blocks a high priority task due to a critical section, sees its priority to be increased to the priority level of the blocked task.
  - $B_i =$ sum of the critical sections of the tasks which have a priority smaller than i.

Shared resource support (5)

- PIP can not be used with more than one shared resource due to deadlock.
- PCP (Priority Ceiling Protocol) [SHA 90] is used instead.
- Implemented in most of real-time operating systems (eg. VxWorks).
- Several implementations of PCP exists : OPCP, ICPP, ...

Shared resource support (6)



- **Ada 2005 implements ICPP (Immediate Ceiling Priority Protocol):**
  - Ceiling priority of a resource = maximum static priority of the tasks which use it.
  - Dynamic task priority = maximum of its own static priority and the ceiling priorities of any resources it has locked.
  - $B_i =$ largest critical section of the tasks sharing the same set of resources than task i.

Shared resource support (7)

- **How to take into account the blocking time B_i with the processor utilization factor test :**
  - Preemptive RM feasibility test:

$$\forall\, i, 1 \leq i \leq n : \sum_{k=1}^{i-1} \frac{C_k}{P_k} + \frac{C_i + B_i}{P_i} \leq i(2^{\frac{1}{i}} - 1)$$

  - Preemptive EDF feasibility test:

$$\forall\, i, 1 \leq i \leq n : \sum_{k=1}^{i-1} \frac{C_k}{P_k} + \frac{C_i + B_i}{P_i} \leq 1$$

Shared resource support (8)

- **How to take into account the blocking time B_i with the worst case response time r_i of the task i (example of a preemptive RM scheduler) :**

$$r_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{P_j} \right\rceil C_j$$

with $hp(i)$ the set of tasks which has a lowest priority than task i.

- Which can be computed by:

$$w_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{P_j} \right\rceil C_j$$

Talk overview

1. **Introduction**
   - Real-time systems.
   - What is real-time scheduling.
   - What we aim to do this afternoon ?
2. **Real-time scheduling theory**
   - Introducing real-time scheduling theory.
   - Usual real-time schedulers.
   - Few words about shared resources.
3. **Ada standards and real-time scheduling**
   - Real-time scheduling with Ada 1995/2005, Ravenscar.
   - POSIX 1003 and its Ada binding.
4. **Scheduling analysis tools and how to use them.**
5. **Summary and further readings**

Real-time scheduling and Ada

- **Real-time scheduling facilities available for Ada practitioners:**
  - ISO/IEC Ada 1995 and 2005 : the Systems Programming Annex C and the Real-Time Annex D [TAF 06].
  - Ada POSIX 1003 Binding [BUR 07, GAL 95].
  - ARINC 653 [ARI 97].
  - ...

Ada 2005 real-time scheduling facilities

- **With Ada 1995/2005, real-time scheduling features are provided by pragmas and specific packages:**

  - How to implement a periodic task:
    1. Representing time ($Ada.Real\_Time$ package).
    2. Implementing periodic release times ($delay$ statement).
    3. Assigning priorities (pragma).
  - How to activate priority inheritance with shared resources (protected objects/types).
  - How to select a scheduler (fixed priority scheduling, EDF, ...).
  - ...

Ada 95/2005: task, time, priorities (1)

```
package Ada.Real_Time is

   type Time is private;
   Time_Unit  : constant := implementation-defined;
   type Time_Span is private;
   ...
   function Clock return Time;
   ...
   function Nanoseconds  (NS : Integer) return Time_Span;
   function Microseconds (US : Integer) return Time_Span;
   function Milliseconds (MS : Integer) return Time_Span;
   function Seconds (S : Integer) return Time_Span;
   function Minutes (M : Integer) return Time_Span;
   ...
```

- $Ada.Real\_Time$ provides a new **monotonic**, **high-resolution** and **documented** "Calendar" package.

Ada 95/2005: task, time, priorities (2)

- $Time$ implements an absolute time. The range of this type shall be sufficient to represent real ranges up to 50 years later.
- $Time\_Span$ represents the length of real-time duration.
- $Time\_Unit$ is the smallest amount of real-time representable by the $Time$ type. It is implementation defined. Shall be less than or equal to 20 microseconds.
- $Clock$ returns the amount of time since $epoch$.
- Some sub-programs which convert input parameters to $Time\_Span$ values (e.g. $Nanoseconds$, $Microseconds$, ...).

Ada 95/2005: task, time, priorities (3)

- **Implementing periodic release times with $delay$ statements:**

1. $delay\ expr$: blocks a task during **at least** $expr$ amount of time.
2. $delay\ until\ expr$: blocks a task until **at least** the particular point in time expressed by $expr$ is reached.

- A task can not be released **before the amount of time** specified with the $delay$ statement.

- But tasks can be released **after the amount of time** specified with the $delay$ statement
- No upper bound on the release time lateness for a $delay$ statement.
- Upper bound lateness shall be documented by the implementation.

Ada 95/2005: task, time, priorities (4)

- **Example of a periodic task (car embedded software example):**

```
with Ada.Real_Time; use Ada.Real_Time;
 ...
 task Tspeed is
 end Tspeed;

 task body Tspeed is
    Next_Time : Ada.Real_Time.Time := Clock;
    Period : constant Time_Span := Milliseconds (250);
 begin
    loop
       --  Read the car speed sensor
       ...
       Next_Time := Next_Time + Period;
       delay until Next_Time;
    end loop;
 end Tspeed;
```

- Use $delay\ until$ instead of $delay$ (due to clock cumulative drift).

Ada 95/2005: task, time, priorities (5)

- **Ada priority model :**

```
package System is

   --  Priority-related Declarations (RM D.1)
   Max_Priority           : constant Positive := 30;
   Max_Interrupt_Priority : constant Positive := 31;

   subtype Any_Priority       is Integer      range  0 .. 31;
   subtype Priority           is Any_Priority range  0 .. 30;
   subtype Interrupt_Priority is Any_Priority range 31 .. 31;

   Default_Priority : constant Priority := 15;
   ...
```

  - **Base** priority : statically assigned.
  - **Active** priority : inherited (rendez-vous, ICPP).
  - $System.Priority$ must provide at least 30 priority levels (but having more levels is better for real-time scheduling analysis).

Ada 95/2005: task, time, priorities (6)

- **Task base priority assignment rules with Ada 1995/2005:**
  - Priority pragma can be used in task specifications.
  - Priority pragma can be assigned to main procedures.
  - Any task without Priority pragma has a priority equal to the task that created it.
  - Any task has a default priority value (see the $System$ package).

```
task Tspeed is
   pragma Priority (10);
end Tspeed;

task Tspeed (My_Priority : System.Priority) is
   entry Service( ...
   pragma Priority (My_Priority);
end Tspeed;
```

Ada 95/2005: task, time, priorities (7)

- Ada 2005 supports several priority inheritance protocols : ICPP (Immediate Ceiling Priority Protocol) and PLCP (Preemption Level Control Protocol).

- Assignment of an ICPP ceiling priority to a protected object/type:

```
protected A_Mutex is
   pragma Priority(15);
   entry E ...
   procedure P...
end A_Mutex;
```

- How to activate priority inheritance with ICPP:

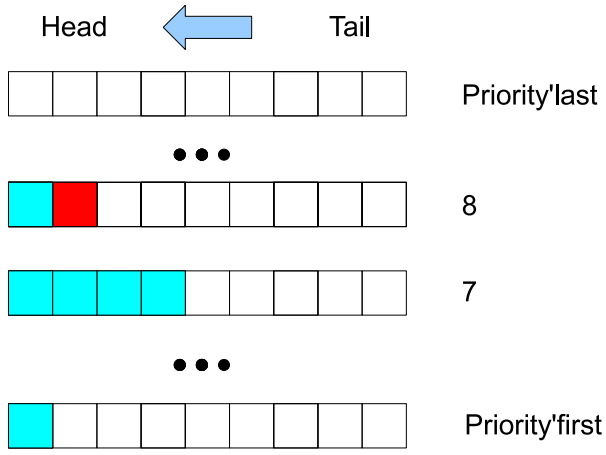
```
pragma Locking_Policy(Ceiling_Locking);
```

Ada 2005 real-time scheduling facilities

- **With Ada 1995/2005, real-time scheduling features are provided by pragmas and specific packages:**

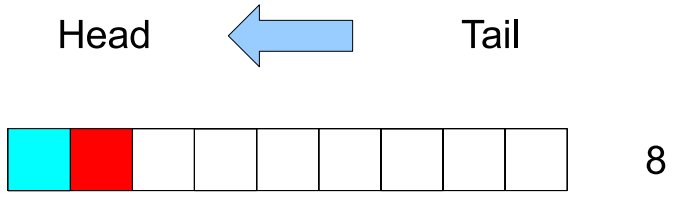
  - How to implement a periodic task:
    1. Representing time ($Ada.Real\_Time$ package).
    2. Implementing periodic release times ($delay$ statement).
    3. Assigning priorities (pragma).
  - How to activate priority inheritance with shared resources (protected objects/types).
  - How to select a scheduler (fixed priority scheduling, EDF, ...).
  - ...

Ada 95/2005 scheduling model (1)



- **Ada 2005 real-time scheduling model:**
  - A queue for each priority level. All ready tasks which have the same active priority level are put in the same queue.
  - Each queue has a dispatching policy.
  - Two-levels of scheduling:
    1. Choose the highest priority queue with at least one ready task.
    2. Choose the task to run of the queue selected in (1), according to the queue dispatching policy.

Ada 95/2005 scheduling model (2)



- Example of the preemptive $FIFO\_Within\_Priorities$ dispatching policy:
  - When a task becomes ready, it is inserted in the tail of its corresponding priority queue.
  - The task at the head of the queue gets the processor when it becomes the highest ready priority task/queue.
  - When a running task becomes blocked or terminated, it leaves the queue and the next task in the queue gets the processor.

$\Longrightarrow$ **We can easily apply fixed priority scheduling feasibility tests if all tasks have different priority levels.**

Ada 95/2005 scheduling model (3)

- **The $FIFO\_Within\_Priorities$ dispatching policy is activated by:**

```
pragma Task_Dispatching_Policy(FIFO_Within_Priorities);
```

- **Ada 2005 also provides other dispatching policies:**

1. Non preemptive fixed priority dispatching:

```
pragma Task_Dispatching_Policy(
   Non_Preemptive_FIFO_Within_Priorities);
```

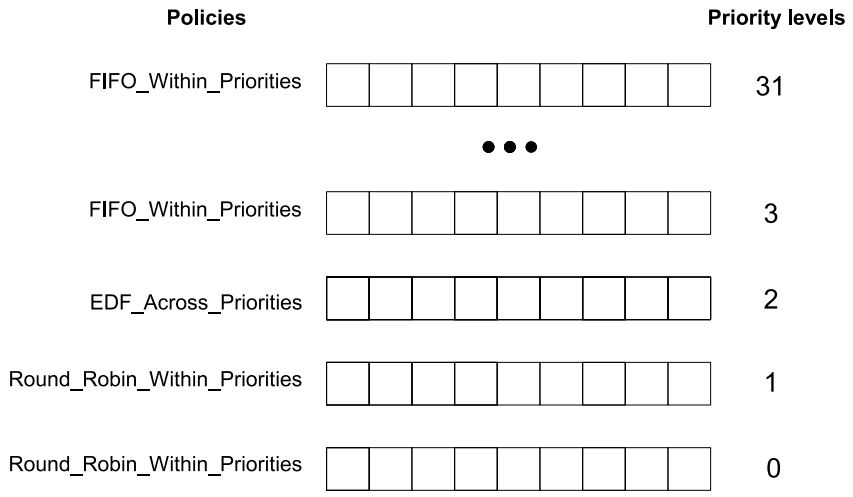
2. Earliest deadline first dispatching:

```
pragma Task_Dispatching_Policy(
   EDF_Across_Priorities);
```

3. Round robin dispatching:

```
pragma Task_Dispatching_Policy(
   Round_Robin_Within_Priorities);
```

Ada 95/2005 scheduling model (4)



- **Ada 2005 allows a program to use different dispatching policies.**

Each priority level may have its own dispatching protocol:

```
pragma Priority_Specific_Dispatching(
    FIFO_Within_Priorities, 3, 31);
pragma Priority_Specific_Dispatching(
    EDF_Across_Priorities, 2, 2);
pragma Priority_Specific_Dispatching(
    Round_Robin_Within_Priorities, 0, 1);
```

Ada 95/2005 scheduling model (5)

- **Example of the "car embedded software example":**

```
package Ada_Tasks is

   task Tdisplay is
      -- Period=100; Capacity=20
      pragma Priority(12);
   end Tdisplay;

   task Tspeed is
      -- Period=250; Capacity=50
      pragma Priority(11);
   end Tspeed;

   task Tengine is
      -- Period=500; Capacity=150
      pragma Priority(10);
   end Tengine;

   ...
```

Ada 95/2005 scheduling model (6)

```
package body Ada_Tasks is

   task body Tspeed is
      Next_Time : Ada.Real_Time.Time := Clock;
      Period : constant Time_Span := Milliseconds (250);
   begin
      loop
         -- Do the job
         Next_Time := Next_Time + Period;
         delay until Next_Time;
      end loop;
   end Tspeed;

   task body Tengine is ...
   task body Tdisplay is ...
end Ada_Tasks;

-- Content of gnat.adc (if compiled with GNAT)
pragma Task_Dispatching_Policy(FIFO_Within_Priorities);
pragma Locking_Policy(Ceiling_Locking);
```

Ada 2005 Ravenscar profile (1)

- **Remember the previously feasibility tests examples:** processor utilization factor test, worst case response time.
- **Each feasibility test has several applicability assumptions.** Processor utilization factor test assumes:
  - Fixed preemptive scheduling.
  - Rate monotonic priority assignment.
  - ICCP shared resources/protected object.
  - Periodic release times.
  - Critical instant.
  - ...
- **How to be sure that your applications is compliant with those feasibility tests assumptions ?**
- **How to increase compliance of your applications with feasibility tests ?** $\Longrightarrow$ use Ravenscar.

Ada 2005 Ravenscar profile (2)

- **What is Ravenscar:**
  - Ravenscar defines an Ada sub-language which is compliant with Rate Monotonic feasibility tests.
  - Ravenscar is a profile which is part of the Ada 2005 standard.
  - A profile is a set of restrictions a program must meet.
  - Restrictions are expressed with pragmas. They are checked at compile-time to enforce the restrictions at execution time.

Ada 2005 Ravenscar profile (3)

- The Ravenscar profile is activated by:

```
pragma profile(Ravenscar);
```

- Examples of the restrictions enforced by Ravenscar:

```
-- Use preemptive fixed priority scheduling
pragma Task_Dispatching_Policy(FIFO_Within_Priorities);

--  Use ICPP
pragma Locking_Policy(Ceiling_Locking);

pragma Restrictions(
  No_Task_Allocators,  -- No task dynamic allocation
                       -- ASSUMPTION RELATED TO  TASK
                       -- THE CRITICAL INSTANT
  No_Dependence => Ada.Calendar,   -- Use Real-time calendar only
  No_Relative_Delay,  -- Disallow time drifting due to
                      -- the use of the delay statement
  ...
  );
```

Real-time scheduling and Ada

- **Real-time scheduling facilities available for Ada practitioners:**
  - ISO/IEC Ada 1995 and 2005 : the Systems Programming Annex C and the Real-Time Annex D [TAF 06].
  - Ada POSIX 1003 Binding [BUR 07, GAL 95].
  - ARINC 653 [ARI 97].
  - ...

POSIX 1003 standard (1)

- Define a standardized interface of an operating system simitar to UNIX [VAH 96].

- Published by ISO and IEEE. Organized in chapters:

Chapters	Meaning
POSIX 1003.1	System Application Program Interface (eg. $fork$, $exec$)
POSIX 1003.2	Shell and utilities (eg. sh)
POSIX 1003.1b [GAL 95]	Real-time extensions.
POSIX 1003.1c [GAL 95]	Threads
POSIX 1003.5	Ada POSIX binding
...	

- Each chapter provides a set of services. A service may be mandatory or optionnal.

POSIX 1003 standard (2)

- Example of operating systems providing 1003.1b : Lynx/OS, VxWorks, Solaris, Linux, QNX, etc .. (actually, most of real-time operating systems).
- POSIX 1003.1b services :

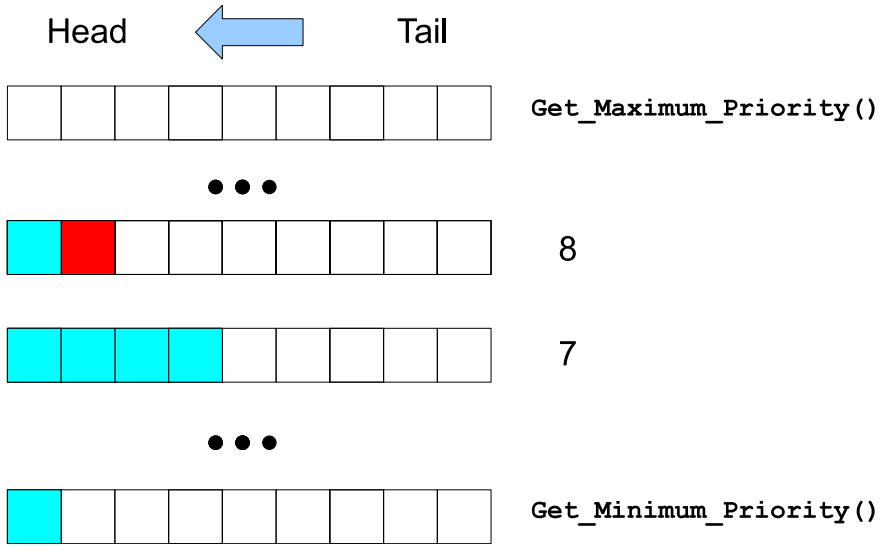
Name	Meaning
_POSIX_PRIORITY_SCHEDULING	Fixed priority scheduling
_POSIX_REALTIME_SIGNALS	Real-time signals
_POSIX_ASYNCHRONOUS_IO	Asynchronous I/O
_POSIX_TIMERS	WatchDogs
_POSIX_SEMAPHORES	Synchronization tools
...	

POSIX 1003 standard (3)

- How the Ada programmer can run POSIX 1003.1b applications ? POSIX 1003.5 Ada binding (e.g. Florist).

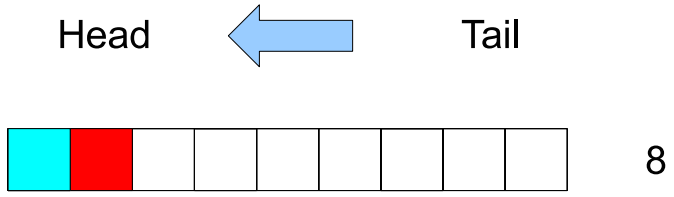
- **This Ada binding provides access to POSIX 1003:**
  - Scheduling services for fixed priority scheduling, EDF, ...
  - Timers to implement periodic release times.

POSIX 1003 standard (4)



- **POSIX real-time scheduling model:**
  - Preemptive fixed priority scheduling. At least 32 priority levels.
  - Two-levels scheduling :
    1. Choose the queue which has the highest priority level with at least one ready process/thread.
    2. Choose a process/thread from the queue selected in (1) according to a **policy**.

POSIX 1003 standard (5)



- **POSIX policies:**
  1. $SCHED\_FIFO$: similar to the $FIFO\_Within\_Priorities$. Ready processes/threads of a given priority level get the processor according to their order in the queue.
  2. $SCHED\_RR$: $SCHED\_FIFO$ with a time quantum. A time quantum is a maximum duration that a process/thread can run on the processor before preemption by an other process/thread of the same queue. When the quantum is exhausted, the preempted process/thread is moved to the tail of the queue.
  3. $SCHED\_OTHER$: implementation defined (usually implements a time sharing scheduler).

POSIX 1003 standard (6)

- **Example of the $Process\_Scheduling$ package which defines:**
  - Priority/policy types.
  - Sub-programs to adapt POSIX application to RTOS features.
  - Sub-programs to change scheduling properties of processes.

```
package POSIX.Process_Scheduling is

   subtype Scheduling_Priority is Integer;

   type Scheduling_Policy is new Integer;
   Sched_FIFO  : constant Scheduling_Policy := ...
   Sched_RR    : constant Scheduling_Policy := ...
   Sched_Other : constant Scheduling_Policy := ...

   type Scheduling_Parameters is private;
```

POSIX 1003 standard (7)

- **Sub-programs which allow the application to adapt itself to the underlying real-time operating system:**

```
package POSIX.Process_Scheduling is
   ...
 function Get_Maximum_Priority (Policy:Scheduling_Policy)
     return Scheduling_Priority;
 function Get_Minimum_Priority (Policy:Scheduling_Policy)
     return Scheduling_Priority;

 function Get_Round_Robin_Interval
     (Process : POSIX_Process_Identification.Process_ID)
     return POSIX.Timespec;
   ...
```

POSIX 1003 standard (8)

- **Set or get policy/priority of a process:**

```
package POSIX.Process_Scheduling is

   procedure Set_Priority
     (Parameters : in out Scheduling_Parameters;
      Priority   : Scheduling_Priority);
   procedure Set_Scheduling_Policy
     (Process    : POSIX_Process_Identification.Process_ID;
      New_Policy : Scheduling_Policy;
      Parameters : Scheduling_Parameters);
   procedure Set_Scheduling_Parameters
     (Process    : POSIX_Process_Identification.Process_ID;
      Parameters : Scheduling_Parameters);

   function Get_Scheduling_Policy ...
   function Get_Priority ...
   function Get_Scheduling_Parameters ...
```

POSIX 1003 standard (9)

- **Example of the car embedded software example:**

```
with POSIX.Process_Identification; use POSIX.Process_Identification;
with POSIX.Process_Scheduling; use POSIX.Process_Scheduling;

   Pid1 : Process_ID;
   Sched1 : Scheduling_Parameters;

begin
   Pid1:=Get_Process_Id;

   Sched1:=Get_Scheduling_Parameters(Pid1);
   Put_Line("Current priority/policy = "
            & Integer'Image(Get_Priority(Sched1))
            & Integer'Image(Get_Scheduling_Policy(Pid1)));

   Set_Priority(Sched1, 10);
   Set_Scheduling_Policy(Pid1, SCHED_FIFO, Sched1);
   Set_Scheduling_Parameters(Pid1, Sched1);
```

POSIX 1003 standard (10)

- **Does an Ada programmer should use POSIX Ada binding ?**
- **Nice sides of POSIX:**
  - POSIX is supported by a large number of RTOS.
  - Analysis with feasibility tests can be performed with the POSIX scheduling framework.
- **But POSIX also has some drawbacks:**
  - What is a POSIX process ? a POSIX thread ? a task ?
  - Programs may be more complex (timers to implement periodic task releases, use of scheduling services).
  - No Ravenscar to handle feasibility test assumptions.
  - Does POSIX really portable since many services are optionnal ?

Real-time scheduling and Ada

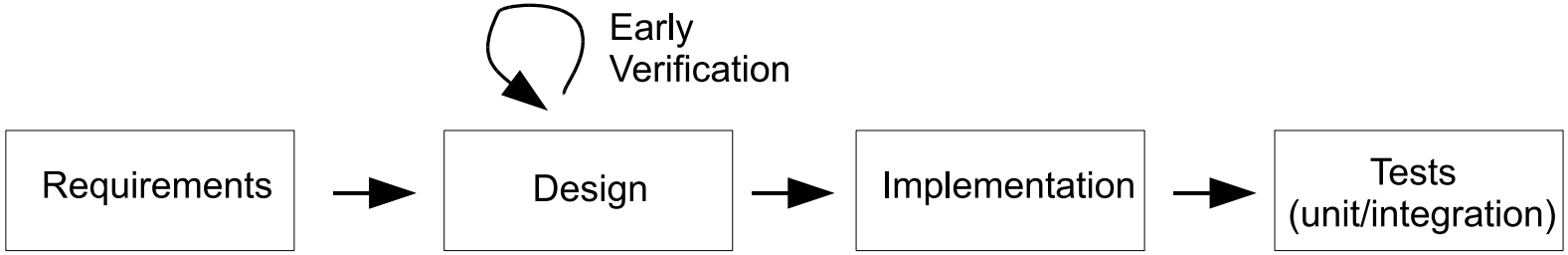
- **Some Ada projects/tools providing Ada 2005 scheduling facilities and/or POSIX Ada binding: :**
  - The Open-Ravenscar project, ORK operating system with Ada 2005 scheduling and POSIX binding. (Universidad Politécnica de Madrid, `http://polaris.dit.upm.es/~ork/`).
  - GNAT GPL, Ada 2005 scheduling and POSIX binding (Florist). (AdaCore, `http://www.adacore.com/`).
  - Marte operating system, implemented with AdaCore GNAT compiler. (Universidad de Cantabria, `http://marte.unican.es/`)

- **Other interesting project providing Ada features which are not compliant with Ada 2005 or POSIX** : RTEMS operating system (OAR Corporation, `http://www.rtems.com/`).

Talk overview

1. **Introduction**
   - Real-time systems.
   - What is real-time scheduling.
   - What we aim to do this afternoon ?
2. **Real-time scheduling theory**
   - Introducing real-time scheduling theory.
   - Usual real-time schedulers.
   - Few words about shared resources.
3. **Ada standards and real-time scheduling**
   - Real-time scheduling with Ada 1995/2005, Ravenscar.
   - POSIX 1003 and its Ada binding.
4. **Scheduling analysis tools and how to use them.**
5. **Summary and further readings**

Scheduling analysis tools (1)



- **Scheduling analysis tools must provide:**
  - A way to model the system architecture to be analyzed.
  - Analysis tools based on:
    1. **Algebraic/analytical tools:** run feasibility tests.
    2. **Model-checking:** compute all reachable states of the system and analyze this set of states.
    3. **Scheduling simulations:** compute scheduling time-lines and analyze them (partial checking).

Scheduling analysis tools (2)

	Feasibility tests	Model checking	Scheduling simulation
Provide proofs	yes	yes	no
Suitable for large system	yes	it depends	it depends
Suitable for specific scheduler/task model	no	yes	yes
Easy to use	yes	no	it depends

- **What kind of tool should we use ?**

Scheduling simulation tools vs feasibility tests tools vs model-checking tools ? All of them ... they are complementary.

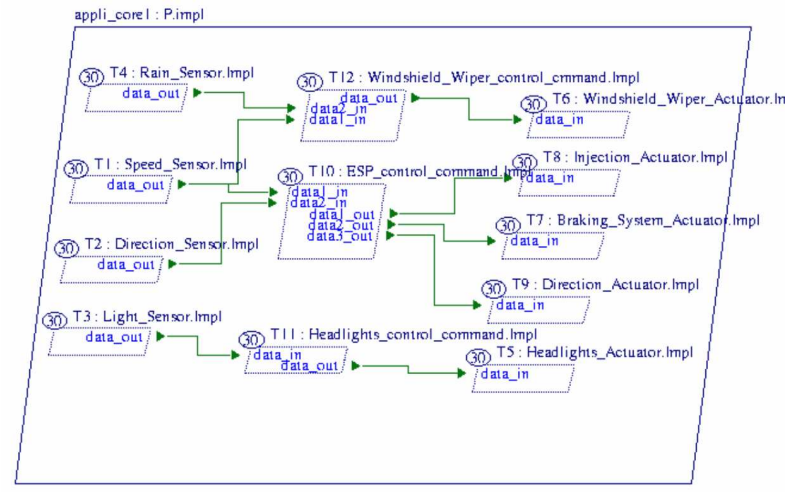
Scheduling analysis tools (3)

- **Examples of both commercial and open-source tools :**
  - MAST (University of Cantabria, `http://mast.unican.es/`).
  - Rapid-RMA (Tri-Pacific Software Inc, `http://www.tripac.com/`).
  - Times (University of Uppsala, `http://www.timestool.com/`)
  - Cheddar (University of Brest and Ellidiss Technologies, `http://beru.univ-brest.fr/~singhoff/cheddar/`).
  - ...
- **There are some tools, but they are not so easy to use :**
  - When and how to use them ?
  - Require deep real-time scheduling analysis theory skills.
  - Relationships with design tools and programming language features.

Scheduling analysis tools (4)

- **Cheddar[SIN 09]** = Ada framework implementing performance analysis tools.
- **Supported architecture design languages :**
  - Support for AADL, PPOOA, UML/Marte design languages.
  - Also provides its own architecture design language which allows the modeling of real-time schedulers.
  - Works with model editors such as Stood (Ellidiss Tech.) or TASTE (ESA/Ellidiss Tech.).
- **Analysis with both feasibility tests and scheduling simulation.**

Scheduling analysis tools (5)



- **Let run some demos:**

1. "Car embedded software" architecture example: 1) edit an AADL model with STOOD 2) run scheduling analysis with Cheddar or AADLInspector.
2. Examples of POSIX schedulings : $SCHED\_RR$ and $SCHED\_FIFO$ policies.
3. Examples of user-defined schedulers : EDF, ARINC 653

Talk overview

1. **Introduction**
   - Real-time systems.
   - What is real-time scheduling.
   - What we aim to do this afternoon ?
2. **Real-time scheduling theory**
   - Introducing real-time scheduling theory.
   - Usual real-time schedulers.
   - Few words about shared resources.
3. **Ada standards and real-time scheduling**
   - Real-time scheduling with Ada 1995/2005, Ravenscar.
   - POSIX 1003 and its Ada binding.
4. **Scheduling analysis tools and how to use them.**
5. **Summary and further readings**

Summary and further readings (1)

- Real-time scheduling theory is a framework to model and analyze critical and non critical real-time multi-tasked systems:
  1. **Provides Schedulability analysis:** either with feasibility tests or scheduling simulation. Some modeling/analysis tools implement them.
  2. **Fixed priority scheduling is supported by most of real-time operating systems.**
  3. **Two standards are available for Ada practitioners : ISO/IEC Ada 2005 (Ravenscar profile) and the POSIX 1003 Ada binding.**

- Feasibility tests presented in this tutorial are (sometimes) extended to be suitable for other schedulers and task models $\Longrightarrow$ see further readings.

Summary and further readings (2)

- **About real-time scheduling theory:**
  - *Scheduling in Real Time Systems*. F. Cottet and J. Delacroix and C. Kaiser and Z. Mammeri. 2002, John Wiley and Sons Ltd editors.
- **Real-time scheduling facilities with POSIX 1003:**
  - *POSIX 4 : Programming for the Real World* . B. O. Gallmeister. O'Reilly and Associates, January 1995.
- **Real-time scheduling facilities with Ada:**
  - *Concurrent and Real Time programming in Ada.* A. Burns and A. Wellings. 2007, Cambridge University Press.
  - *Building Parallel, Embedded, and Real-Time Applications with Ada.* J. W. McCormick, F. Singhoff, J. Hugues. Cambridge University Press, July 2010.

Summary and further readings (3)

- **Other books on real-time systems:**
  - *Real-Time Systems and Programming Languages*. A. Burns. 2009, Addison Wesley; 4th Revised edition.
  - *Real-Time Systems Design and Analysis*. Phillip A. Laplante. 1994, Wiley-IEEE Press.

Summary and further readings (4)

Any questions ?

To contact us:

mccormick@cs.uni.edu,

singhoff@univ-brest.fr

Bibliography (1)

[ARI 97] Arinc. *Avionics Application Software Standard Interface*. The Arinc Committee, January 1997.

[BUR 07] A. Burns and A. Wellings. *Concurrent and Real Time programming in Ada. 2007*. Cambridge University Press, 2007.

[BUT 03] G. Buttazzo. « Rate monotonic vs. EDF: Judgment day ». *n Proc. 3rd ACM International Conference on Embedded Software, Philadephia, USA*, October 2003.

[GAL 95] B. O. Gallmeister. *POSIX 4 : Programming for the Real World* . O'Reilly and Associates, January 1995.

[LIU 73] C. L. Liu and J. W. Layland. « Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment ». *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[SHA 90] L. Sha, R. Rajkumar, and J.P. Lehoczky. « Priority Inheritance Protocols : An Approach to real-time Synchronization ». *IEEE Transactions on computers*, 39(9):1175–1185, 1990.

Bibliography (2)

[SIN 09] F. Singhoff, A. Plantec, P. Dissaux, and J. Legrand. « Investigating the usability of real-time scheduling theory with the Cheddar project ». *Real-Time Systems*, 43(3):259–295, 2009.

[STA 88] John Stankovic. « Misconceptions about real-time computing ». *IEEE Computer*, October 1988.

[TAF 06] S. T. Taft, R. A. Duff, R. L. Brukardt, E. Ploedereder, and P. Leroy. *Ada 2005 Reference Manual. Language and Standard Libraries. International Standard ISO/IEC 8652/1995(E) with Technical Corrigendum 1 and Amendment 1*. LNCS Springer Verlag, number XXII, volume 4348., 2006.

[VAH 96] U. Vahalia. *UNIX Internals : the new frontiers*. Prentice Hall, 1996.