

# AADL : about scheduling analysis



# Scheduling analysis, what is it ?

---

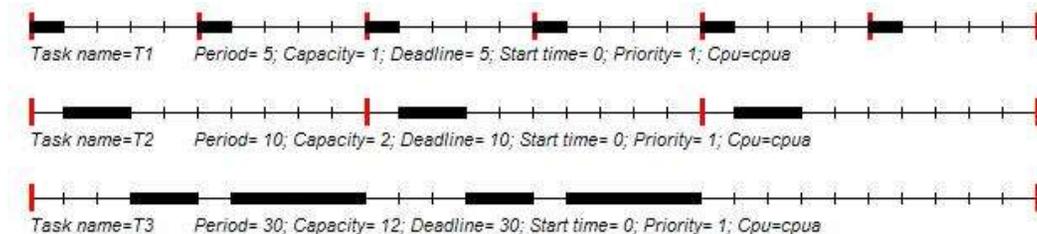
- ❑ **Embedded real-time critical systems** have temporal constraints to meet (e.g. deadline).
- ❑ Many systems are built with operating systems providing multitasking facilities ... Tasks may have deadline.
- ❑ **But, tasks make temporal constraints analysis difficult to do :**
  - ❑ We must take the task scheduling into account in order to check task temporal constraints.
  - ❑ Scheduling (or schedulability) analysis.

# Real-Time scheduling theory

1. **A set of simplified tasks models** (to model functions of the system)
2. **A set of analytical methods** (called feasibility tests)
  - **Example:**

$$R_i \leq \text{Deadline} \quad R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j$$

3. **A set of scheduling algorithms:** build the full scheduling/GANTT diagram



# Real-Time scheduling theory is hard to apply

---

- Real-Time scheduling theory
  - Theoretical results defined from 1974 to 1994:  
feasibility tests exist for uniprocessor architectures
- Now supported at a decent level by POSIX 1003 real-time operating systems, ARINC653, ...
- Industry demanding
  - Yet, hard to use

# Real-Time scheduling theory is hard to apply

---

- Requires strong theoretical knowledge/skills
  - Numerous theoretical results: how to choose the right one ?
  - Numerous assumptions for each result.
  - How to abstract/model a system to verify deadlines?
- How to integrate scheduling analysis in the engineering process ?
  - When to apply it ? What about tools ?

**It is the role of an ADL to hide those details**

# Uniprocessor fixed priority scheduling

---

## □ **Fixed priority scheduling :**

- Scheduling based on fixed priority => priorities do not change during execution time.
- Priorities are assigned at design time (off-line).
- Efficient and simple feasibility tests.
- Scheduler easy to implement into real-time operating systems.

## □ **Rate Monotonic priority assignment :**

- Optimal assignment in the case of fixed priority scheduling and uniprocessor.
- Periodic tasks only.

# Uniprocessor fixed priority scheduling

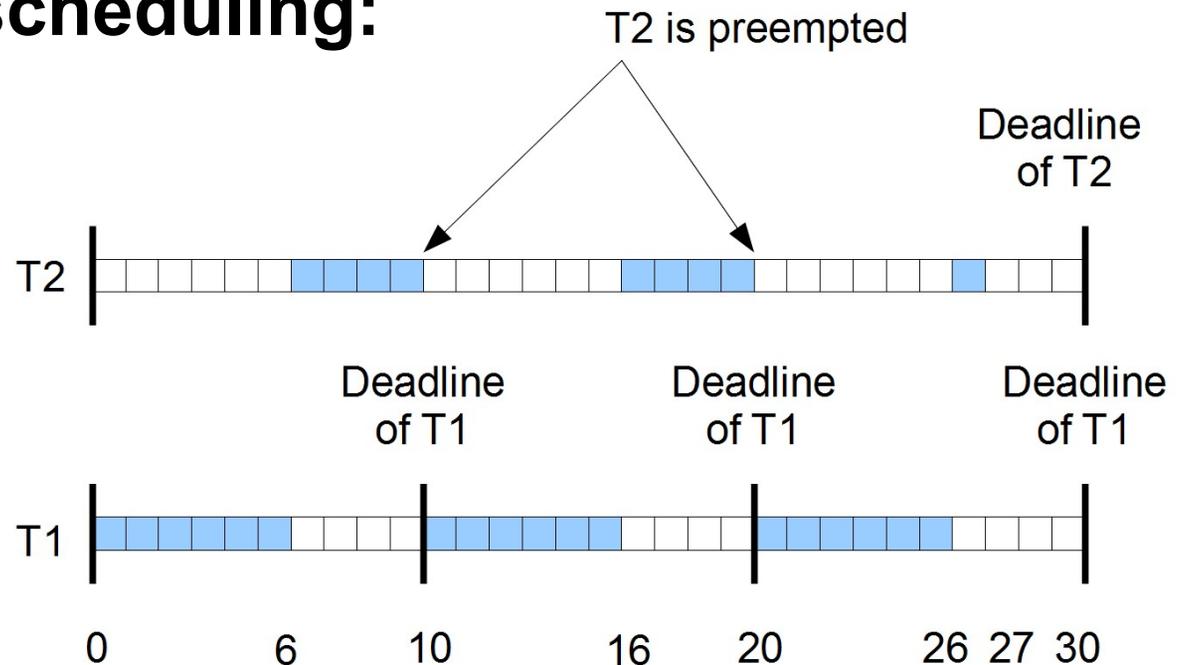
---

## □ Two steps:

1. **Rate monotonic 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.

# Uniprocessor fixed priority scheduling

## □ Rate Monotonic assignment and preemptive fixed priority scheduling:



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

# Uniprocessor fixed priority scheduling

---

## □ **Feasibility/Schedulability tests to predict at design-time if deadline will be met:**

1. **Run simulations on hyperperiod** =  $[0, \text{LCM}(P_i)]$ . Sufficient and necessary condition.
2. **Processor utilization factor test:**  
$$U = \sum_{i=1}^n C_i/P_i \leq n \cdot (2^{\frac{1}{n}} - 1) \quad (\text{about } 69\%)$$

Rate Monotonic assignment and preemptive scheduling.  
Sufficient but not necessary condition.
3. **Task worst case response time, noted  $R_i$**  : delay between task release time and task completion time. Any priority assignment but preemptive scheduling.

# Uniprocessor fixed priority scheduling

---

## □ Compute $R_i$ , task $i$ worst case response time:

- Task  $i$  response time = task  $i$  capacity + delay the task  $i$  has to wait for higher priority task  $j$ . Or:

$$R_i = C_i + \sum_{j \in hp(i)} \text{waiting time due to } j \quad \text{or} \quad R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j$$

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

# Uniprocessor fixed priority scheduling

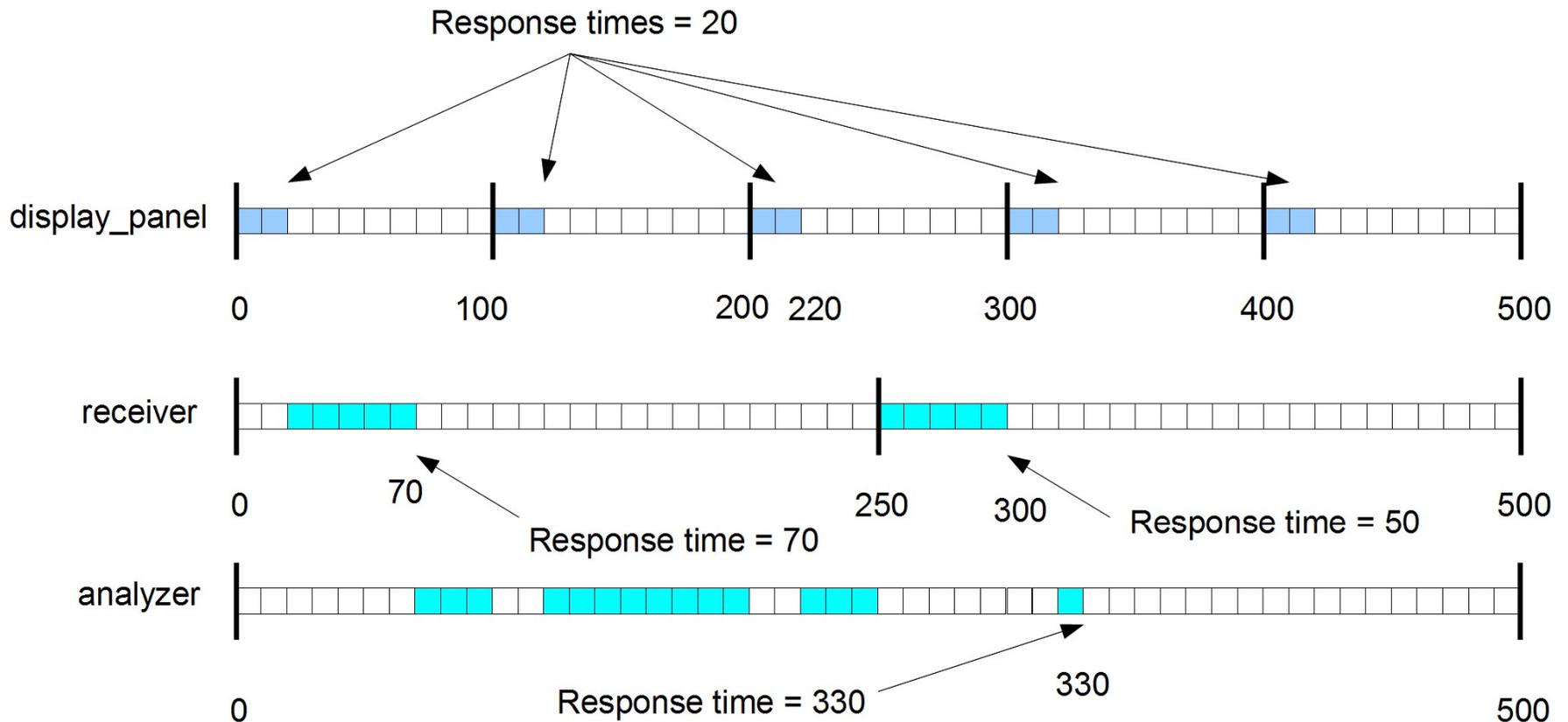
---

- To compute task response time: compute  $wi^k$  with:

$$wi^n = Ci + \sum_{j \in hp(i)} \lceil wi^{n-1} / Pj \rceil \cdot Cj$$

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

# Uniprocessor fixed priority scheduling



# Fixed priority and shared resources

---

- Previous tasks were independent ... does not really exist in true life.
  
- **Task dependencies :**
  - Shared resources.
    - E.g. with AADL: threads may wait for AADL protected data component access.
  - Precedencies between tasks.
    - E.g with AADL: threads exchange data by data port connections.

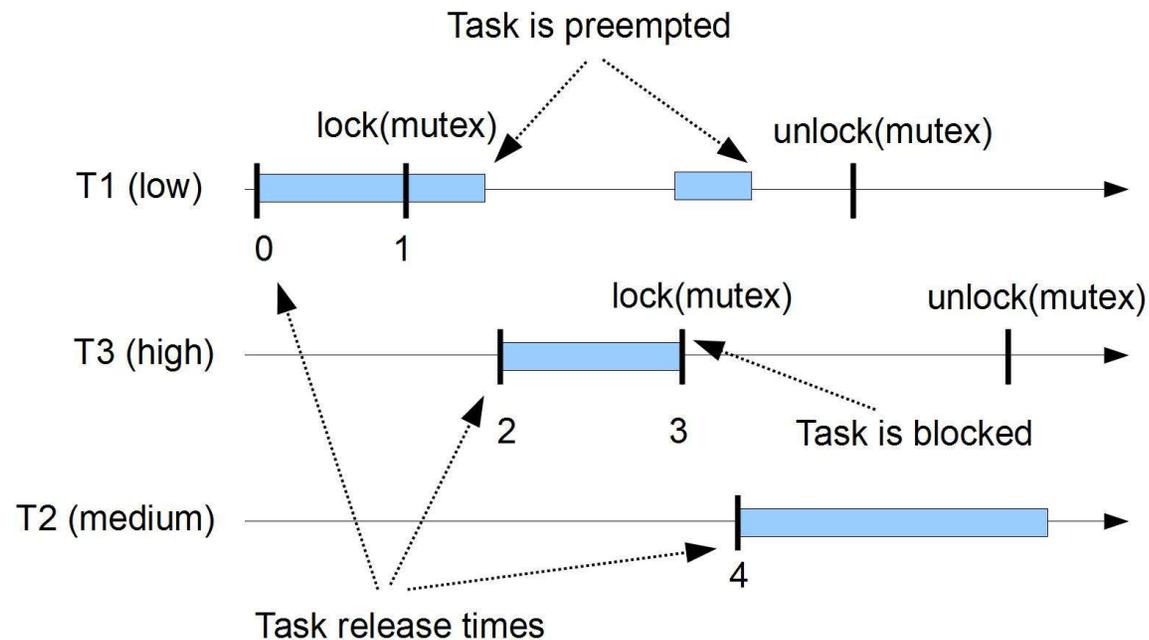
# Fixed priority and shared resources

---

- Shared resources are modeled by semaphores for scheduling analysis.
- **We use specific semaphores implementing inheritance protocols:**
  - To take care of priority inversion.
  - To compute worst case task waiting time for the access to a shared resource. Blocking time  $B_i$ .
- **Inheritance protocols:**
  - PIP (Priority inheritance protocol), can not be used with more than one shared resource due to deadlock.
  - PCP (Priority Ceiling Protocol) , implemented in most of real-time operating systems (e.g. VxWorks).
  - Several implementations of PCP exists: OPCP, ICPP, ...

# Fixed priority and shared resources

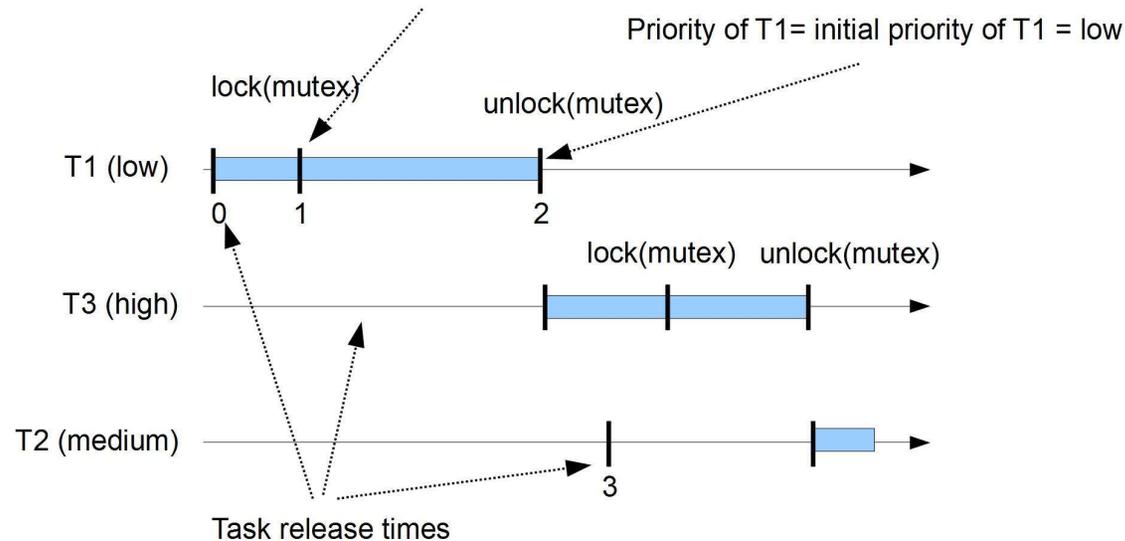
- **What is Priority inversion:** a low priority task blocks a high priority task



- $B_i$  = worst case on the shared resource waiting time.

# Fixed priority and shared resources

Priority of T1= ceiling priority of « mutex » = high



## □ ICPP (Immediate Ceiling Priority Protocol):

- Ceiling priority of a resource = maximum fixed priority of the tasks which use it.
- Dynamic task priority = maximum of its own fixed priority and the ceiling priorities of any resources it has locked.
- $B_i$ =longest critical section ; prevent deadlocks

# Fixed priority and shared resources

---

## □ How to take into account the waiting time $B_i$ :

- Processor utilization factor test :

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

- Worst case response time :

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

# AADL to the rescue ?

---

- **Issues when we try to apply scheduling analysis:**
  - Many scheduling feasibility tests, many assumptions
  - Ensure model elements are compliant with analysis/feasibility test requirements/assumptions
  - Ensure all required model elements are given for the analysis
  
- **AADL helps for the first issue:**
  - AADL as a pivot language between tools. International standard.
  - Close to the real-time scheduling theory: real-time scheduling analysis concepts can be found. Ex:
    - Component categories: thread, data, processor
    - **Property:** Deadline, Fixed Priority, ICPP, Ceiling Priority, ...

# Property sets for scheduling analysis

---

## □ Properties related to processor component:

```
Preemptive_Scheduler : aadlboolean applies to (processor);
```

```
Scheduling_Protocol:
```

```
  inherit list of Supported_Scheduling_Protocols  
  applies to (virtual processor, processor);
```

```
-- RATE_MONOTONIC_PROTOCOL,
```

```
-- POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL, ..
```

# Property sets for scheduling analysis

---

## □ Properties related to the threads/data components:

```
Compute_Execution_Time: Time_Range  
  applies to (thread, subprogram, ...);
```

```
Deadline: inherit Time => Period applies to (thread, ...);
```

```
Period: inherit Time applies to (thread, ...);
```

```
Dispatch_Protocol: Supported_Dispatch_Protocols  
  applies to (thread);  
  -- Periodic, Sporadic, Timed, Hybrid, Aperiodic, Background,  
  ...
```

```
Priority: inherit aadlinteger applies to (thread, ..., data);
```

```
Concurrency_Control_Protocol:  
  Supported_Concurrency_Control_Protocols applies to (data);  
  -- None, PCP, ICPP, ...
```

# Property sets for scheduling analysis

---

## □ Example:

**thread implementation** receiver.impl

**properties**

```
Dispatch_Protocol => Periodic;
Compute_Execution_Time => 31 ms .. 50 ms;
Deadline => 250 ms;
Period => 250 ms;
Priority => 5;
```

**end** receiver.impl;

**data implementation** target\_position.impl

**properties**

```
Concurrency_Control_Protocol
=> PRIORITY_CEILING_PROTOCOL;
```

**end** target\_position.impl;

**process implementation** processing.others

**subcomponents**

```
receiver : thread receiver.impl;
analyzer : thread analyzer.impl;
target : data target_position.impl;
...
```

**processor implementation** leon2

**properties**

```
Scheduling_Protocol =>
    RATE_MONOTONIC_PROTOCOL;
Preemptive_Scheduler => true;
```

**end** leon2;

**system implementation** radar.simple

**subcomponents**

```
main : process processing.others;
cpu : processor leon2;
```

...

# Cheddar : a framework to assess schedulability of AADL models

---

- **Cheddar tool** = analysis framework (queueing system theory & real-time scheduling theory)
  - + internal ADL (architecture description language)
  - + various standard ADL parsers (AADL, MARTE UML)
  - + simple model editor
  - + ...
  
- **Two versions** :
  - Open source (Cheddar) : educational and research, with OSATE or TASTE
  - Commercial product (AADL Inspector) : Ellidiss Tech product.
  
- **Supports** : Ellidiss Tech., Conseil régional de Bretagne, Brest Métropole, EGIDE/Campus France, H2020 MOSAR, Thales DMS & Com, BPI France, Université de Bretagne Occidentale, CNES

# Cheddar : a framework to assess schedulability of AADL models

## Demos:

- Scheduling analysis of the radar example with Cheddar ... and with AADL Inspector also

